Xbox® Hardware Hacking

In Review

S-BOXes and Xboxes

Andrew "bunnie" Huang, PhD
bunnie@xenatera.com

Acknowledgements

- I didn’t do this by myself:
  - Andy “numbnut” Green, Michael “mist” Steil, Milosch Meriac, Franz Lehrer, Jeff “koitsu” Mears, xor, adq, luc, head, visor, roastbeef, kgasper, xerox, lordvictory, pixel8, siliconice, caustik…and others I cannot name ☺

Outline

- Background
- History of the first hardware hacks
- Summary of security
- Later hacks
- Future possibilities

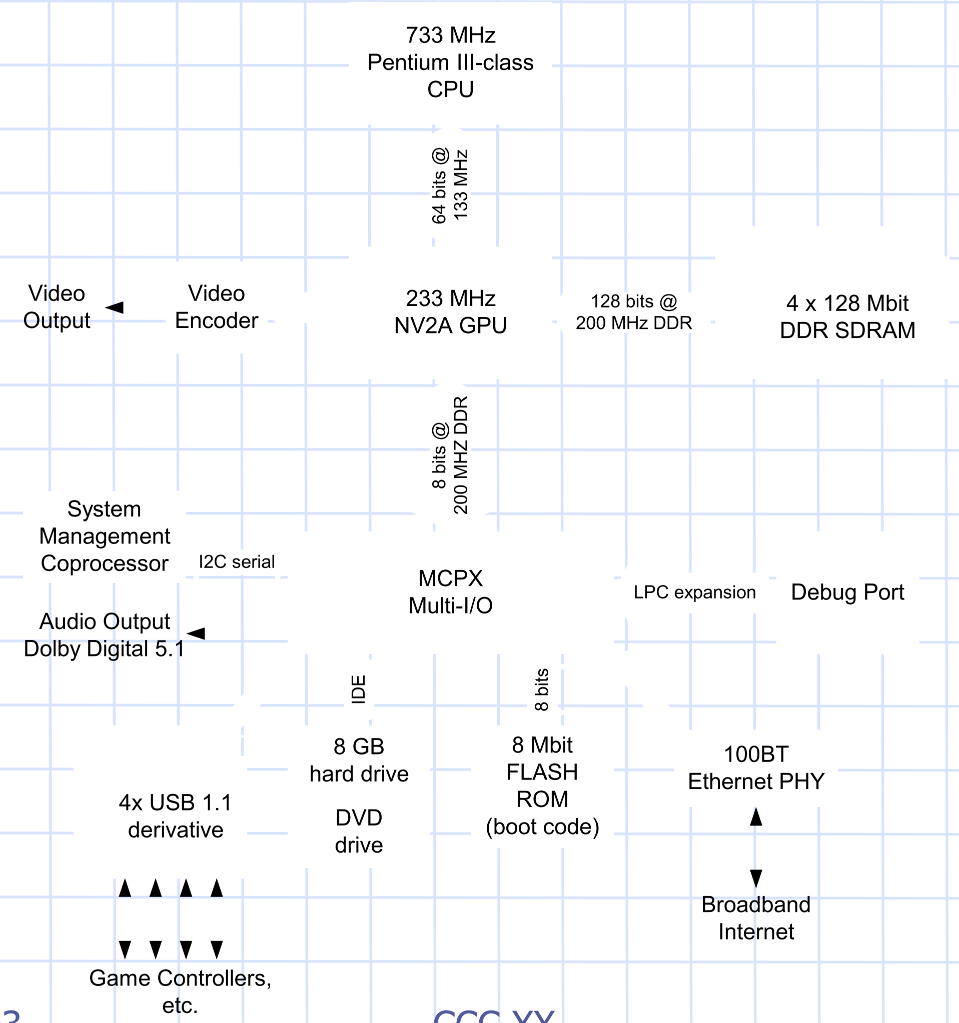
More History than News

- Many have heard about Xbox hardware hacks
- Xbox hardware has changed little since its introduction
  - At 19C3, Andy presented most of the significant/latest facts

What is an Xbox?

- Xbox is an embedded PC
  - 733 MHz Intel Pentium III-class processor
  - nVidia nForce-derivative chipset
  - 64 MB DDR SDRAM
  - 100 Base-T ethernet port
  - VGA graphics capability
  - USB ports
  - 10 GB IDE hard drive
  - IDE DVD-ROM drive

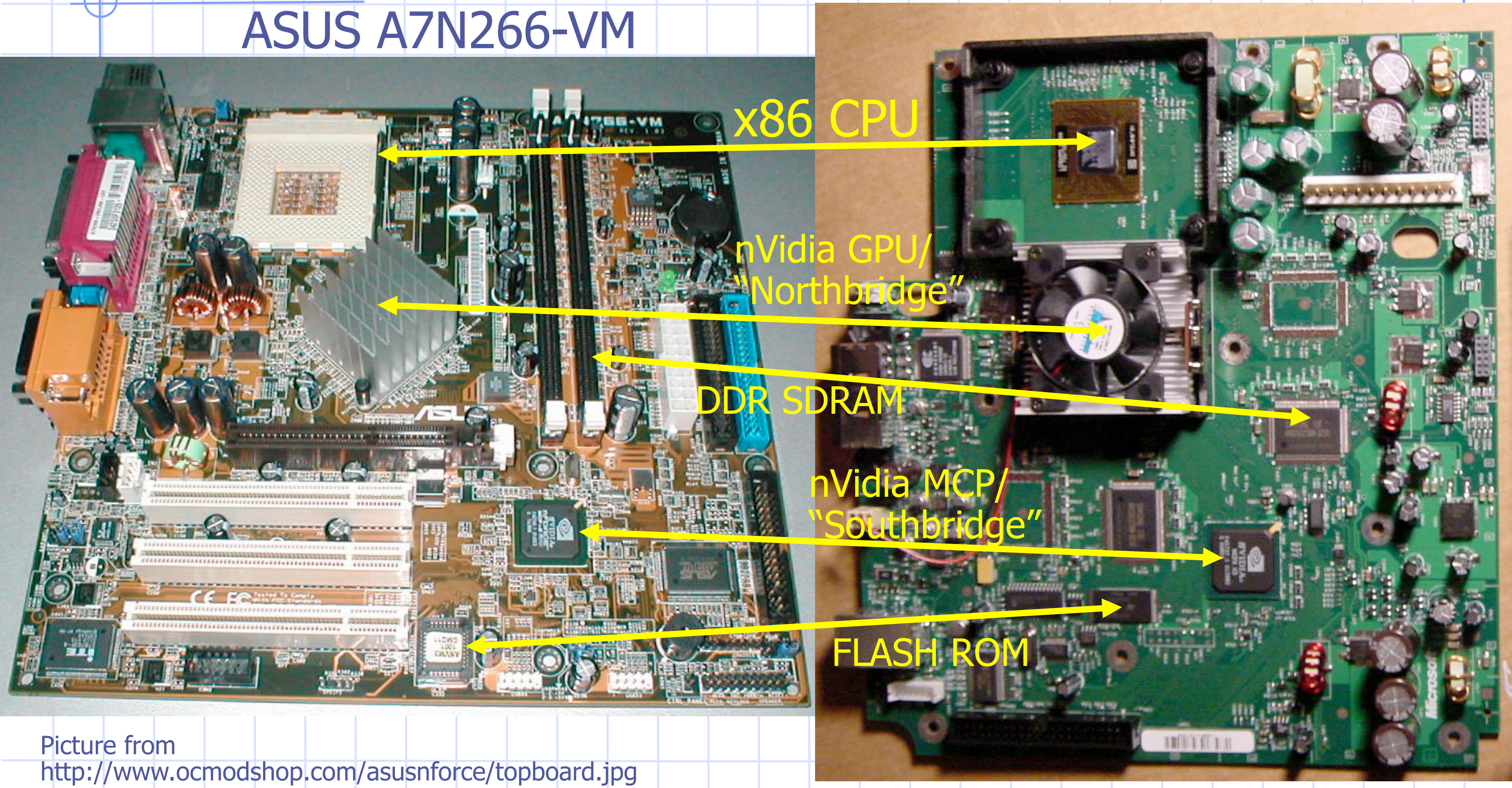
Xbox Internal Block Diagram



Comparison to Stock PC Hardware

ASUS A7N266-VM

Xbox Motherboard



x86 CPU

nVidia GPU/ "Northbridge"

DDR SDRAM

nVidia MCP/ "Southbridge"

FLASH ROM

Picture from
http://www.ocmodshop.com/asusnforce/topboard.jpg

What’s Different

- No PS/2 kbd, mouse
- No parallel or serial ports
- No expansion slots
- Intelligent system management controller
- Modified system ASICs

Security Rationale: Economics

- Hardware is sold at a loss
  - "Loss Leader"
  - Make up the difference in sales of games, services

Economic Details

- At introduction
  - US$100-200 lost per Xbox console
  - Microsoft makes ~$7/title for third-party games
  - Microsoft makes about 3-4x more on first-party titles
  - Sell about 10-20 games to break even
- Originally over US$1000 in software
  - Maybe today the Xbox is faring better; as little as four or five titles may be required to break even

Armchair Economics

- What about subscription royalties?

$50	Xbox Live! Starter kit + 1 yr subs.

Armchair Economics

- What about subscription royalties?

$50	Xbox Live! Starter kit + 1 yr subs.
($15)	retailer’s margin
($10)	operating cost per user (est.)
($20)	depreciation, captitalization
	e.g., investment of US$1 billion

Armchair Economics

- What about subscription royalties?

$50	Xbox Live! Starter kit + 1 yr subs.
($15)	retailer’s margin
($10)	operating cost per user (est.)
($20)	depreciation, captitalization
	e.g., investment of US$1 billion
$5	profit per year offsetting initial hdwe loss

⇒ Assuming initial hdwe sales loss of $100, console will not make money over its operational lifetime

Bottom Line?

- Microsoft wants to be **the** OS in your home!
  - To quote J. Allard, Xbox head honcho: "We [Microsoft] already make a nickel every time you work. Wouldn't it be great if we also made a nickel every time you played?"

Game Consoles

- What’s the big deal?
  - US$31bb market in 2002 (projected)[1]
    - US$22bb for consoles/hardware alone
    - Constant growth throughout 2002 despite downturn
  - Million+ unit/month hardware volumes
  - Widely deployed, high-profile embedded hardware market

[1] Reuters, “Video-game sales to top $31 billion”, June 24 2002

How Much Security?

- Sufficient deterrent to ensure that:
  - O($100) in games, services are purchased over console lifetime
  - On-line gaming experience is enjoyable
    - A billion-dollar investment on Microsoft’s part
    - This may be one of the biggest differentiation points for the Xbox

Security Rationale: Summary

- Prevent the following key scenarios:
  - Game copying
  - Game cheating
    - Ensure an enjoyable on-line gaming experience
  - Emulation
    - Stock PC booting a copied Xbox game
    - Modified PC booting a copied Xbox game
  - Conversion to stock PC
    - Subsidized Windows platform
    - Linux/freeware platform
    - Embedded controller

Outline

- Background
- History of the first hardware hacks
- Summary of security
- Later hacks
- Future possibilities

Why Did I Do It?

- Curiosity
- Challenge
- Fun!

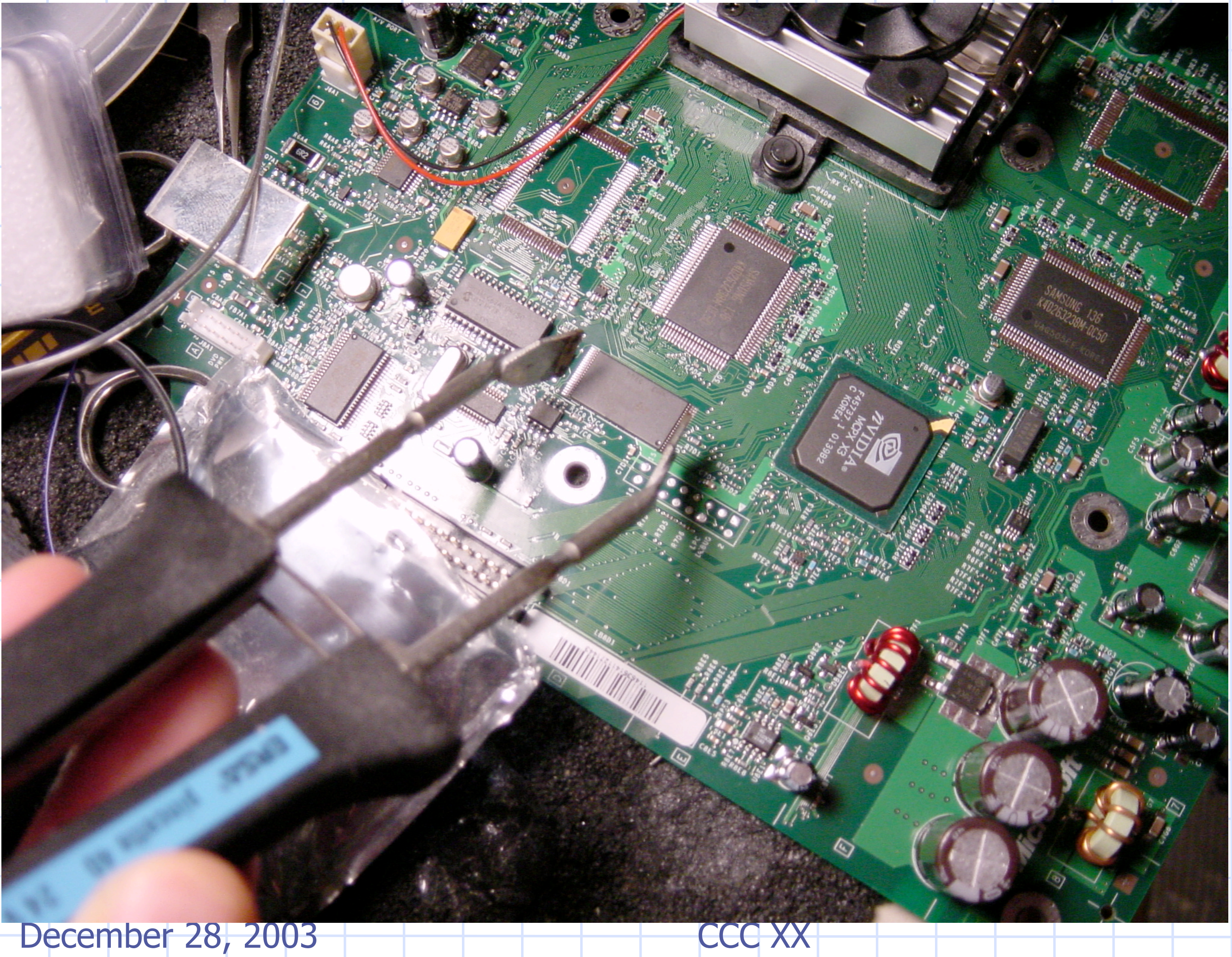
Hackgeschichte

- Xbox was released in November 2002
  - I didn't get mine until late November
  - Nikki got it as a Christmas gift
- Tried obvious/easy things
  - Extract FLASH ROM contents

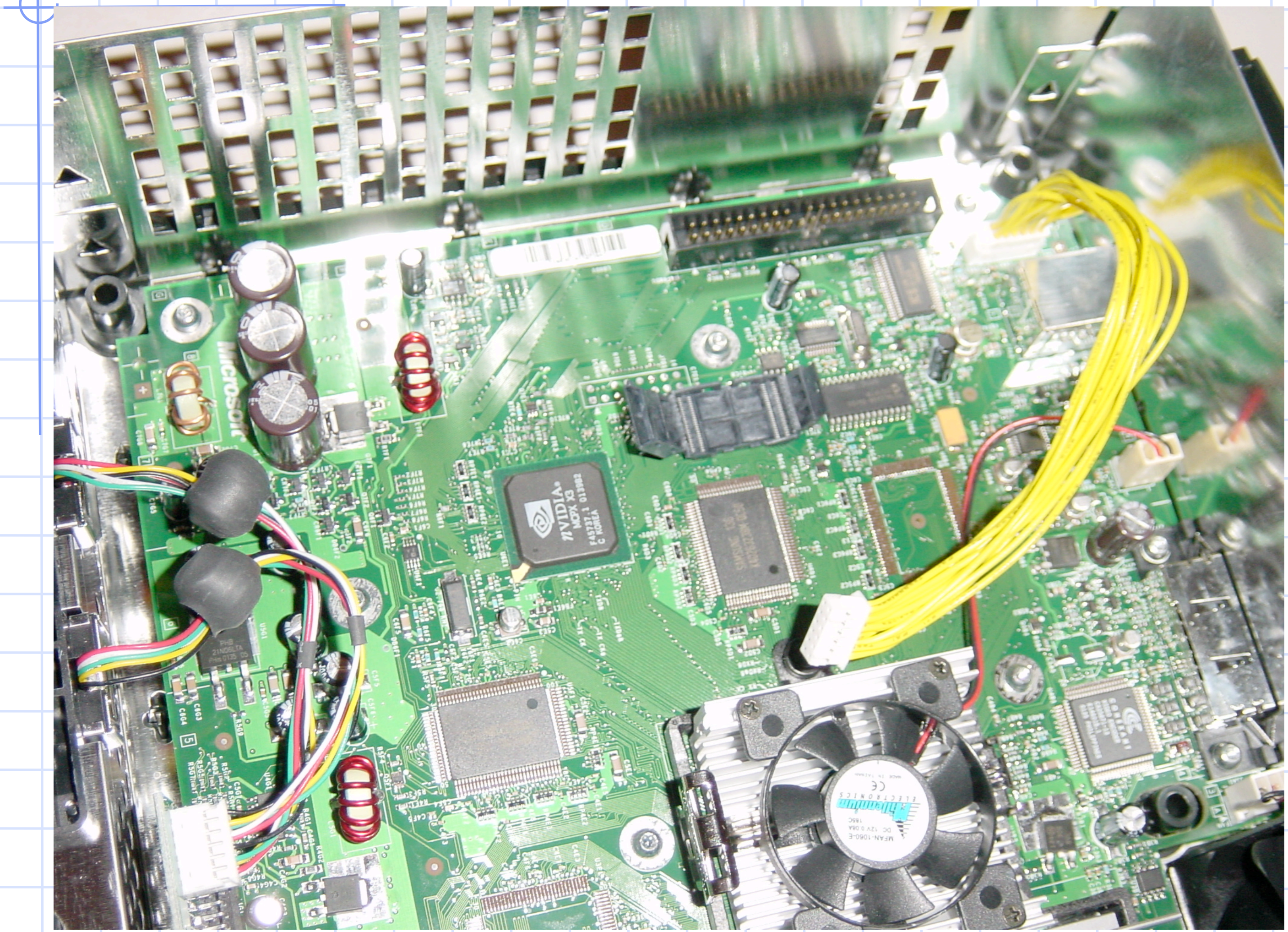
Extracting FLASH

- FLASH is soldered to board
  - Desolder using “tongs” style iron
  - Install cheap TSOP socket
- Other options
  - Use desoldering alloy
  - More difficult to clean up

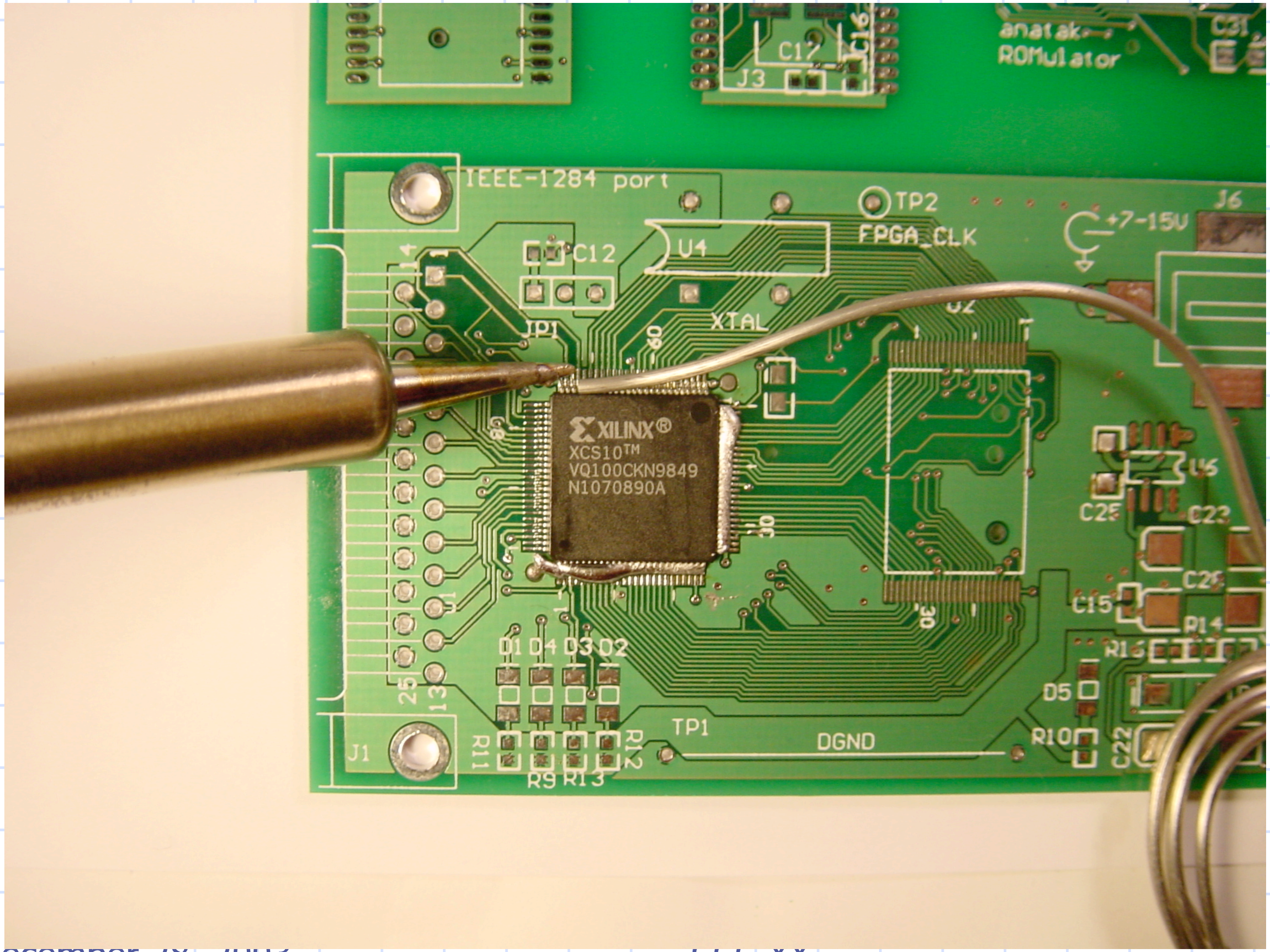
Desoldering



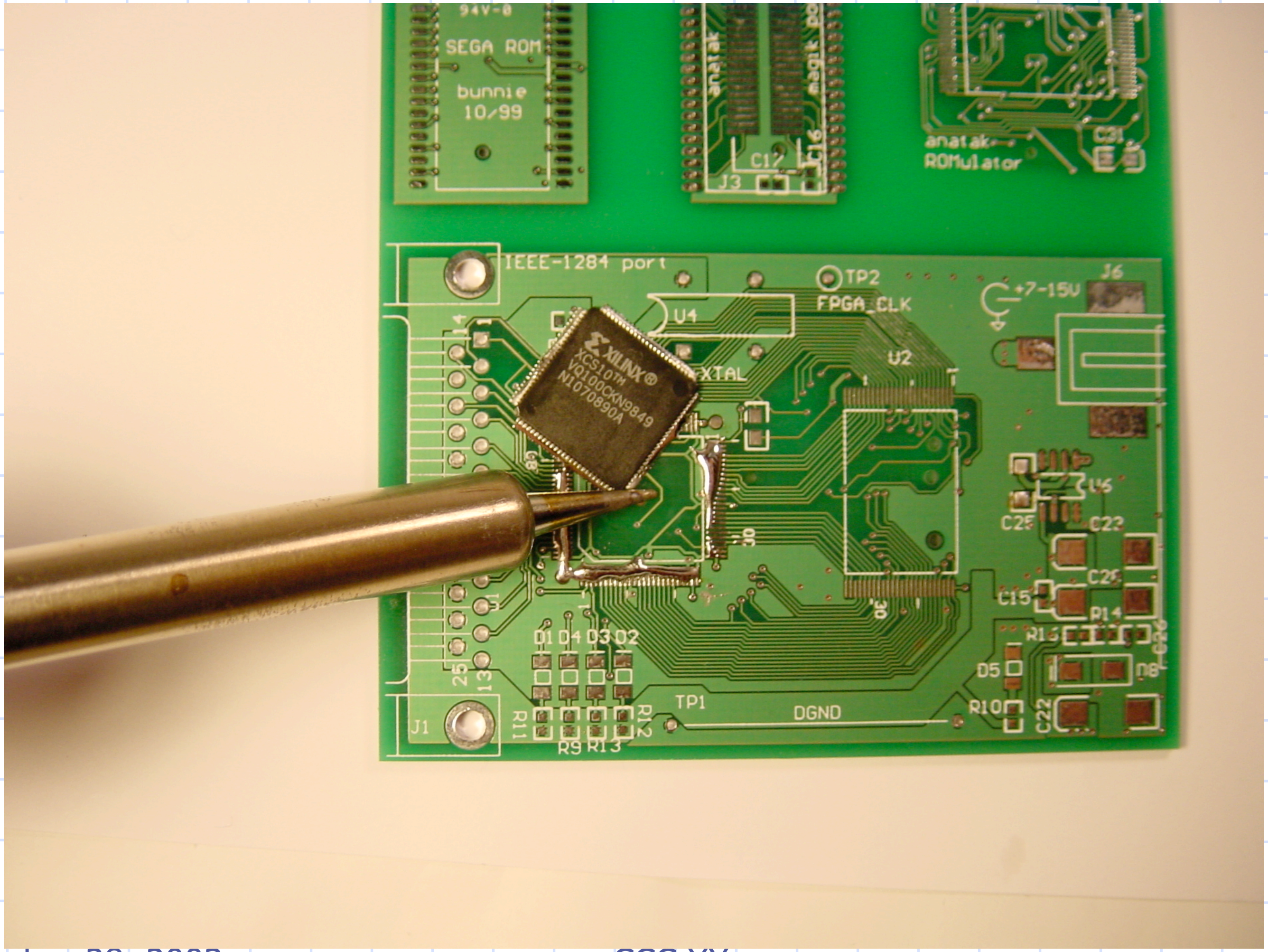
Socketing



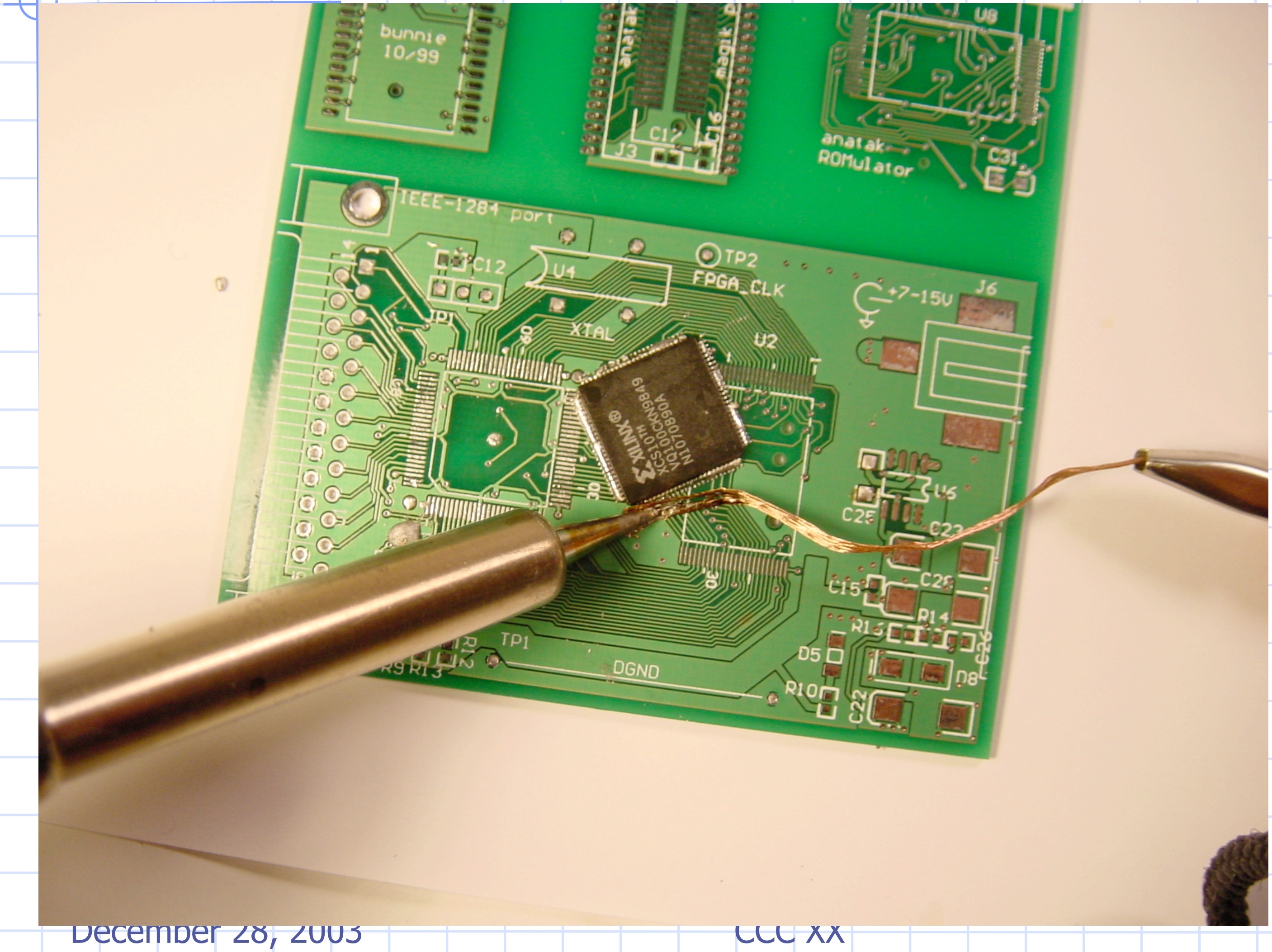
Desoldering



Desoldering



Cleaning Up


bunnie
10/99
anatak
magik
J3
C17
C16
U8
anatak
ROMulator
C31
IEEE-1284 port
TP2
FPGA_CLK
C12
U4
XTAL
U2
+7-15V
J6
XILINX
XCS10CVQ100
U6
C25
C23
C15
C29
R14
R16
D5
R10
C22
D8
TP1
DGND
R9 R13
R12

ROM Analysis

- I soon posted the ROM to my website so others could help me analyze it
  - MSFT called within 12 hours to have me remove the posting
  - No threats of lawsuits, though

ROM Analysis

- ROM contents
  - Primarily code/data, either compressed and/or encrypted
  - At boot vector (0xFFFF.FFF0), plaintext code was found
  - Code performs machine initialization using “jamtables”
  - Then decrypts a block of data and jumps to the code
  - Cipher is RC-4-like
    - I am told that it was a leaked prototype for RC-7
    - Key is located in the memory!

Cipher Listing

```
// key initialization routine
unsigned char K[256]; // 0xFFFFC80 in flash
unsigned char S[256]; // 0x10000 in SDRAM
for( i = 0; i < 256; i++ ) {
  S[i] = i;
}
j = 0;
for( i = 0; i < 256; i++ ) {
  // RC-4 would do j = (j + K[i] + S[i]) % 256
  j = (j + K[i] + S[j]) % 256;
  // swap S[i], S[j]
  temp = S[i];
  S[i] = S[j];
  S[j] = temp;
}
// decryption routine
unsigned char cipherText[16384]; // 0xFFFFA000 in FLASH
unsigned char plainText[16384]; // 0x400000 in SDRAM
for( index = 0x4000, i = 0, k = 0; index > 0; index-- ) {
  // xbox version
  t = (S[i] ^ cipherText[k]) % 256;
  plainText[k] = t;
  // swap( S[i], S[t] );
  temp = S[i];
  S[i] = S[t];
  S[t] = temp;
  i = (i + 1) % 256;
  k++;
}
```

First Conundrum

- In theory:
  - Cipher is symmetric
  - Key is known
  - Should be able to seize control of the machine by making own own encrypted images
  - Should be able to decrypt kernel for analysis
- In practice:
  - Cipher doesn’t seem to work!
  - In fact, machine initialization sequence is all wrong as well!

Theories Fly

- Maybe data/address lines are rotated or scrambled?
- Secondary crypto processor?
- Boot code in CPU?
- Boot code in chipset?

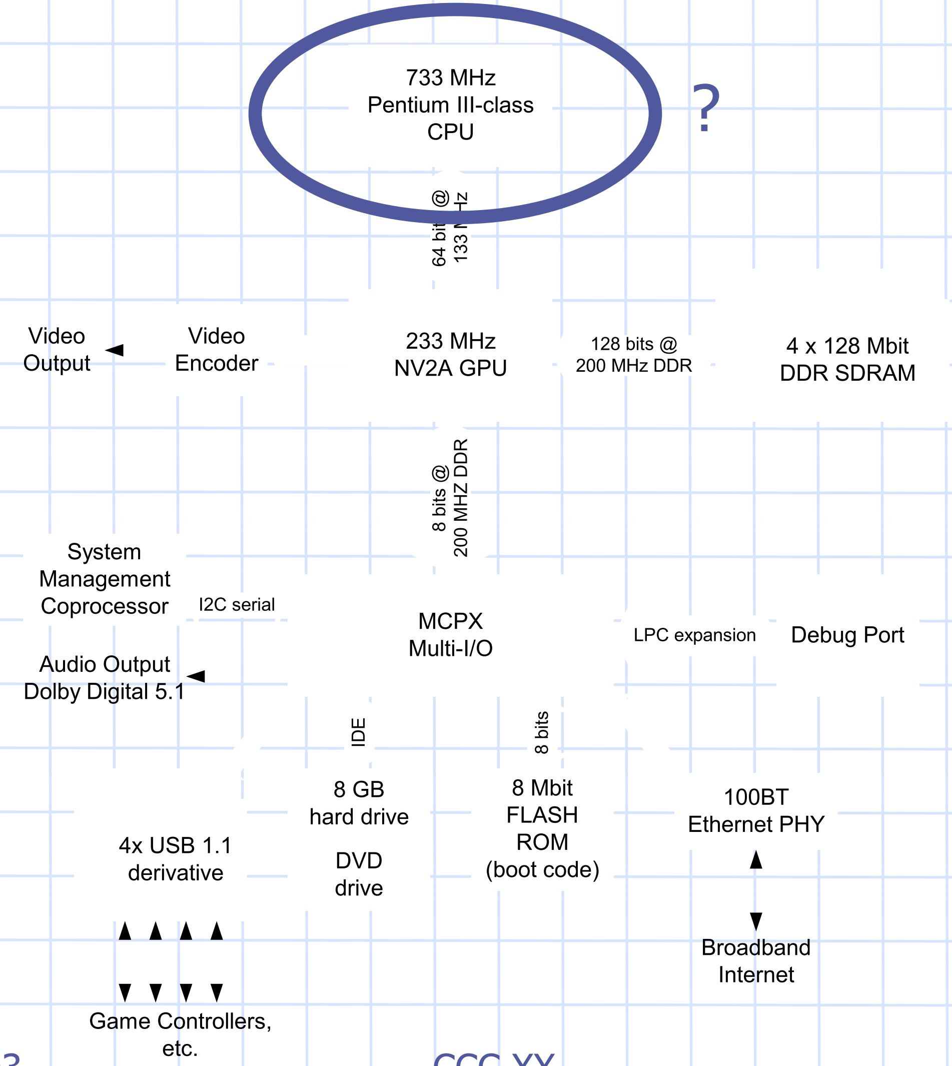
The Crucial Observation

- A friend observed:
  - Changing **just** the boot vector bytes did nothing to the Xbox
  - But changing bytes at random in the body of the ROM crashed the Xbox (generally)

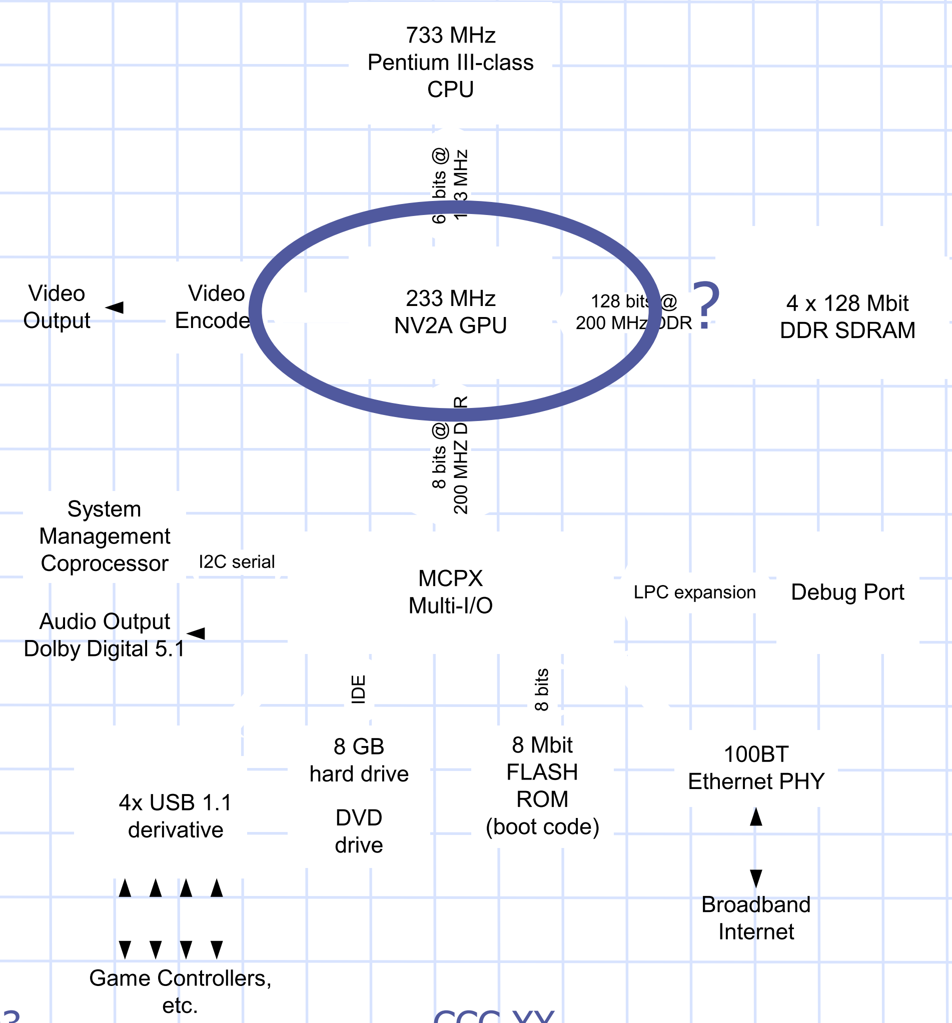
Further Experimentation

- Poking around further around the boot vector revealed about a 512-byte boundary for immunity to changes
- Validated the alternate boot code location

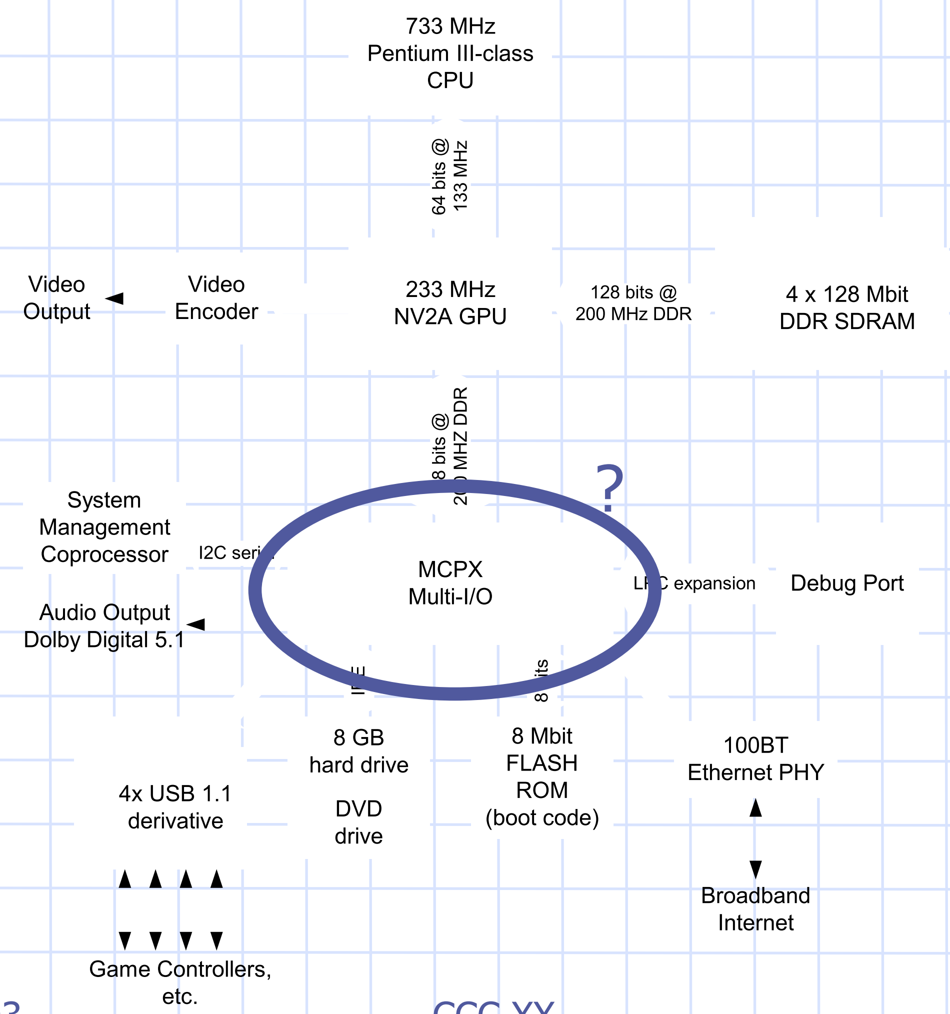
Where Could it Be, and How to Find It?



Where Could it Be, and How to Find It?



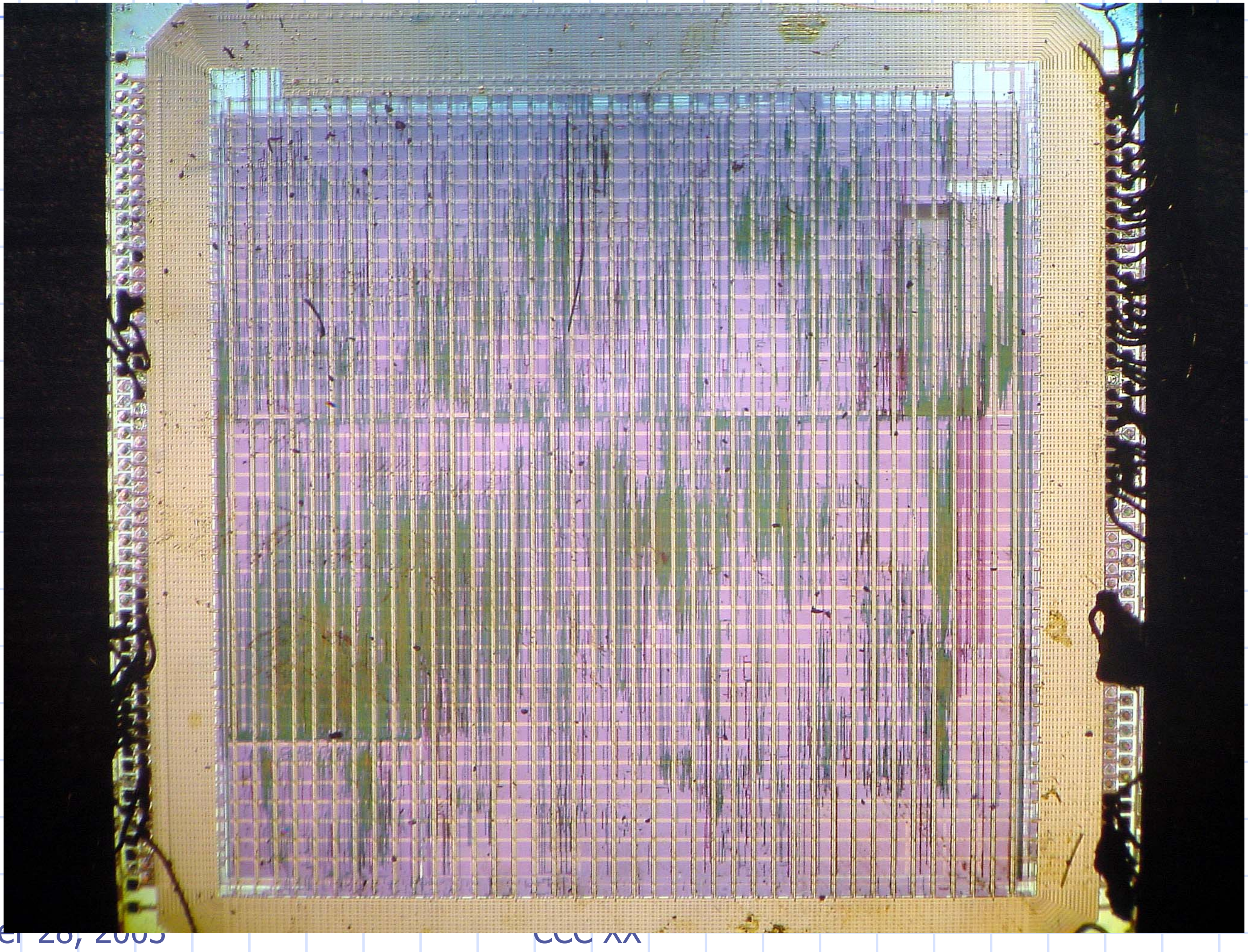
Where Could it Be, and How to Find It?



Approaches

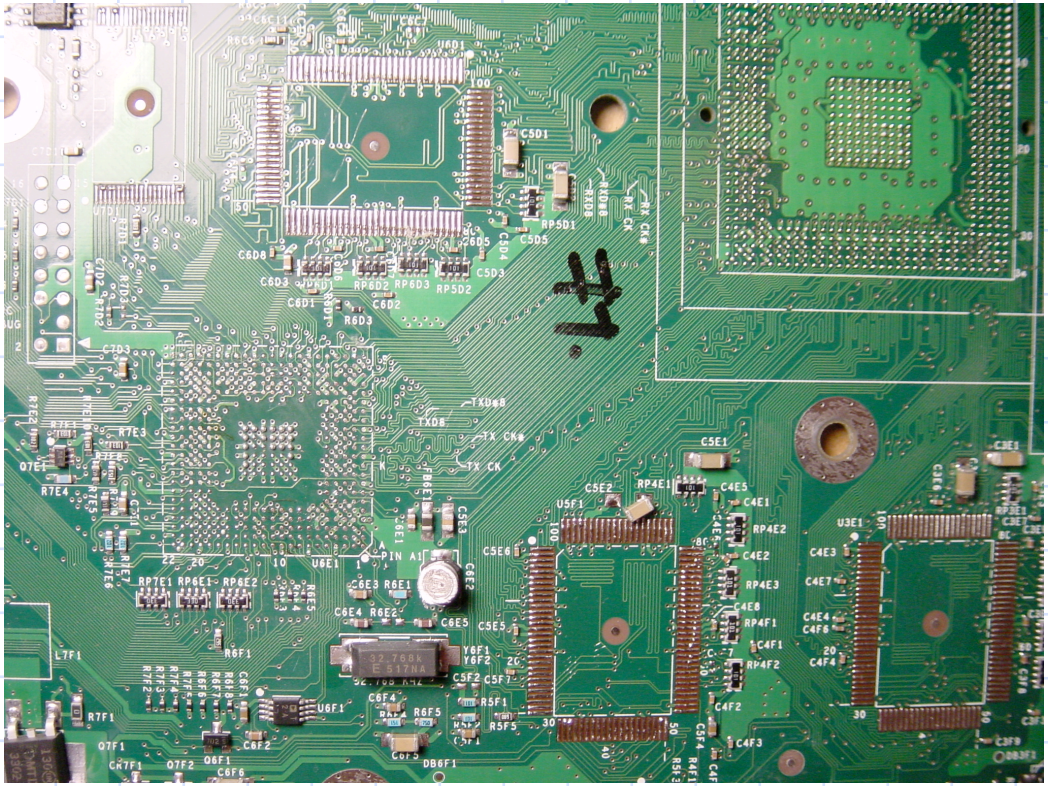
- Physical Inspection
- CPU-ICE
  - Can step through/stop CPU execution
  - Small problem with SMC resetting the system unless deactivated
  - ICE initialization time is longer than reset timeout
- Bus Tapping
  - NB
  - Memory
    - Could be easier b/c can get TSOP100 probes
    - But probably less likely because crypto routine runs in cache
  - SB

Physical Inspection

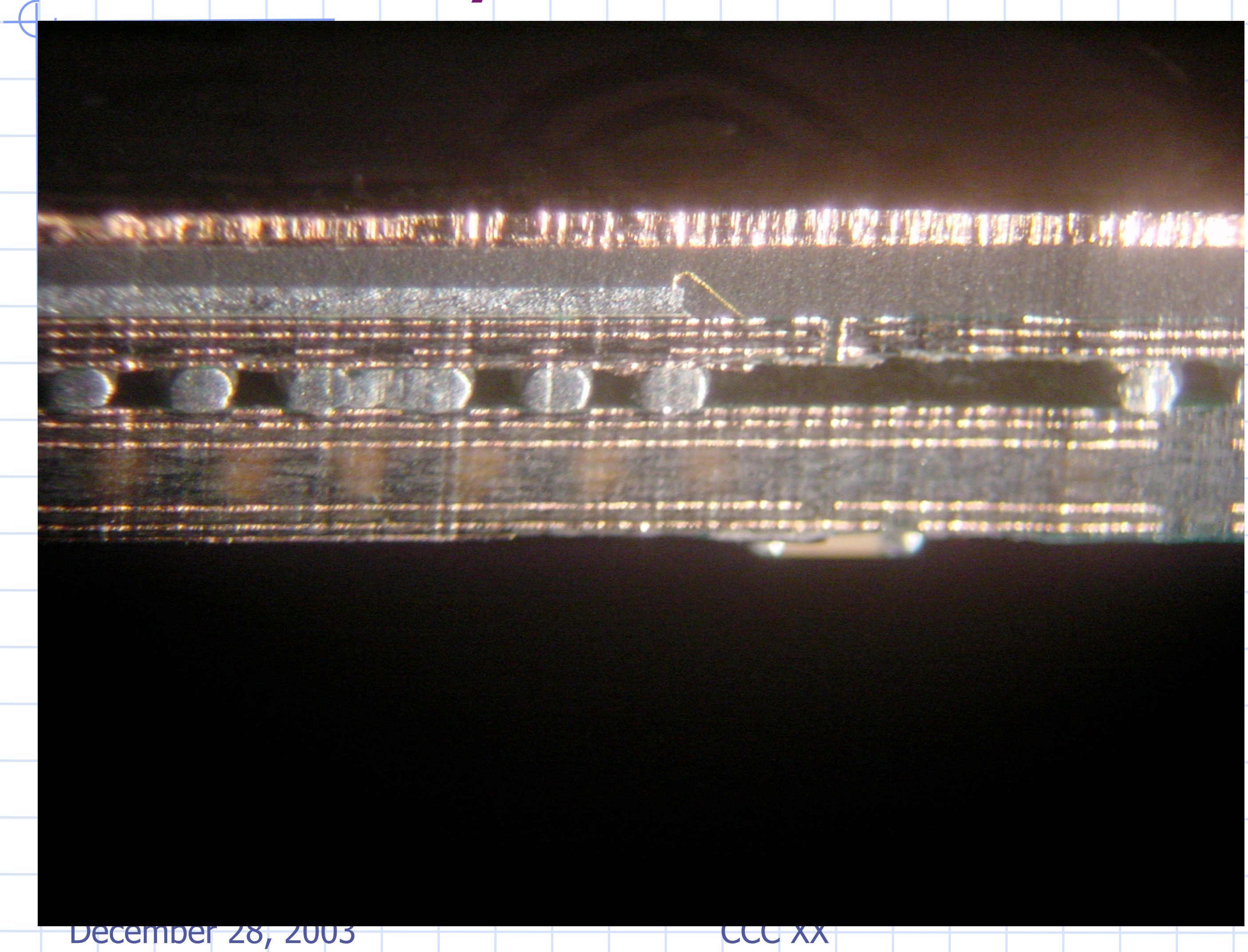


Incidentally…

- It helped a lot to sacrifice an Xbox



Board Layers



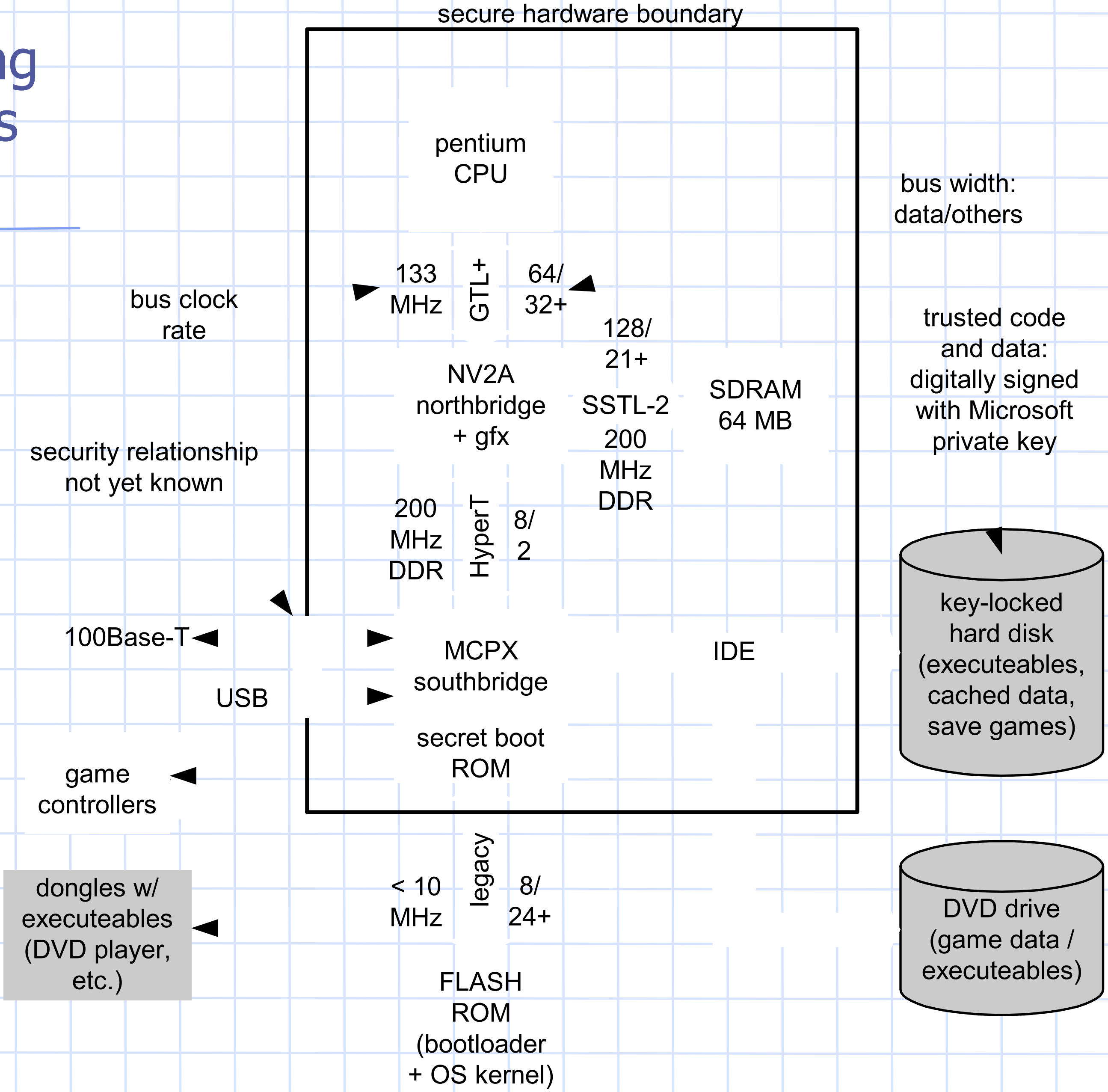
Approaches

- Physical Inspection
- CPU-ICE
  - Can step through/stop CPU execution
  - Small problem with SMC resetting the system unless deactivated
  - ICE initialization time is longer than reset timeout
- Bus Tapping
  - NB
  - Memory
    - Could be easier b/c can get TSOP100 probes
    - But probably less likely because crypto routine runs in cache
  - SB

Approaches

- Physical Inspection
- CPU-ICE
  - Can step through/stop CPU execution
  - Small problem with SMC resetting the system unless deactivated
  - ICE initialization time is longer than reset timeout
- Bus Sniffing
  - NB
  - Memory
    - Could be easier b/c can get TSOP100 probes
    - But probably less likely because crypto routine runs in cache
  - SB

Bus Sniffing Candidates



secure hardware boundary

pentium CPU

133 MHz

GTL+

64/ 32+

NV2A northbridge + gfx

128/ 21+

SSTL-2

200 MHz DDR

SDRAM 64 MB

200 MHz DDR

HyperT

8/ 2

MCPX southbridge

secret boot ROM

IDE

100Base-T

USB

game controllers

dongles w/ executeables (DVD player, etc.)

< 10 MHz

legacy

8/ 24+

FLASH ROM (bootloader + OS kernel)

key-locked hard disk (executeables, cached data, save games)

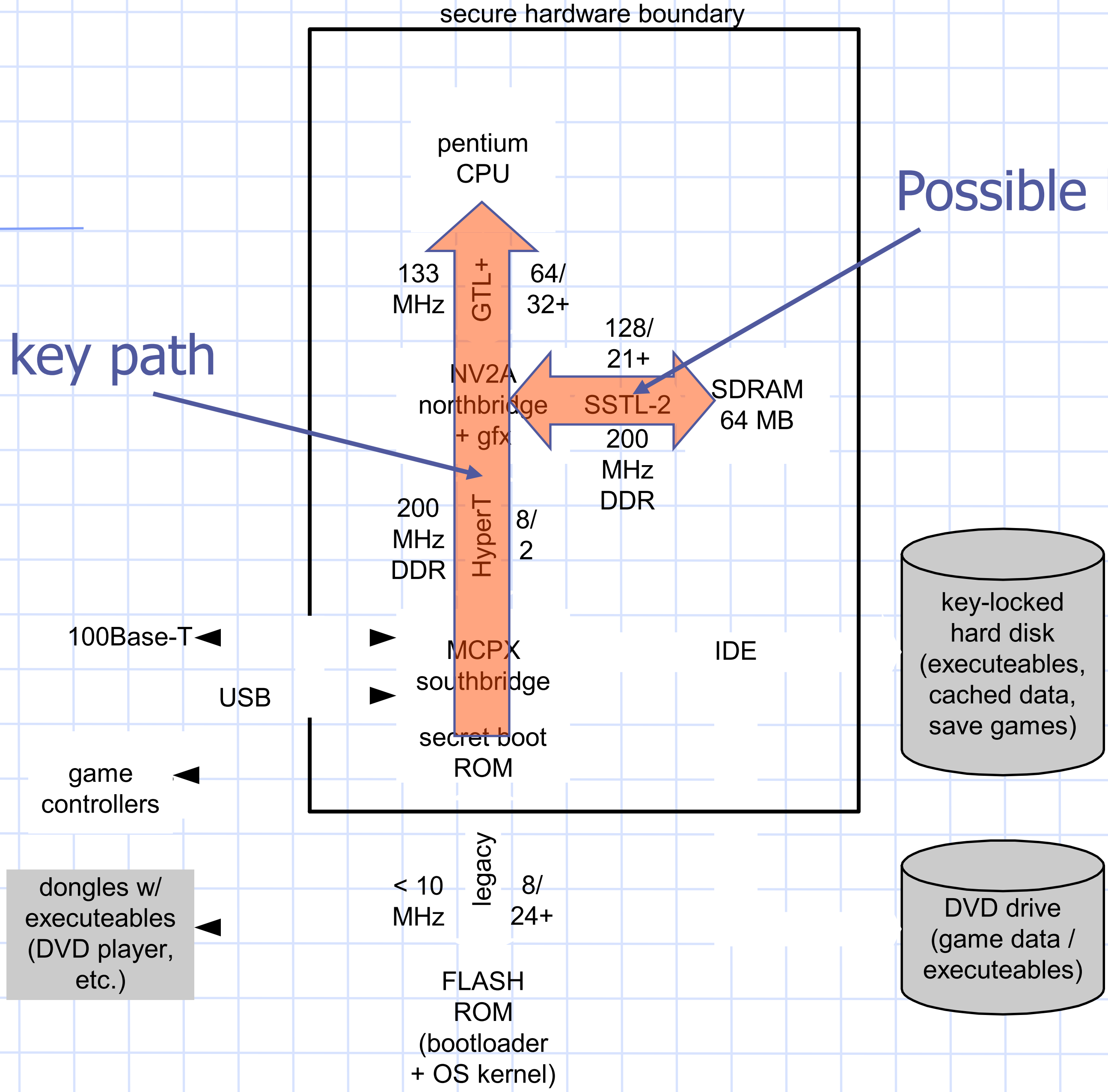
DVD drive (game data / executeables)

bus clock rate

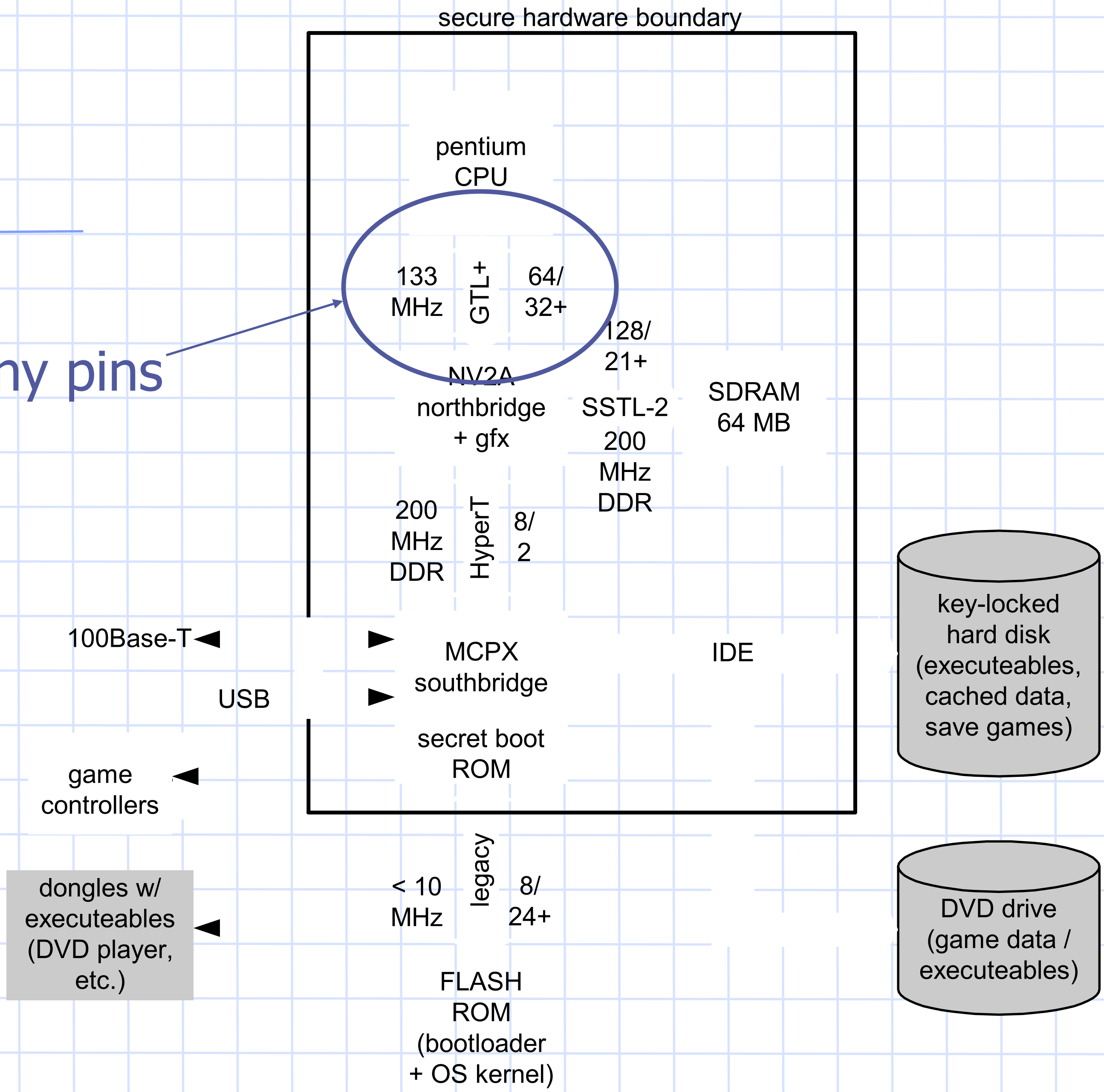
bus width: data/others

security relationship not yet known

trusted code and data: digitally signed with Microsoft private key


secure hardware boundary
pentium CPU
133 MHz
GTL+
64/ 32+
128/ 21+
NV2A northbridge + gfx
SSTL-2
SDRAM 64 MB
200 MHz DDR
200 MHz DDR
HyperT
8/ 2
100Base-T
USB
MCPX southbridge
IDE
secret boot ROM
key-locked hard disk (executeables, cached data, save games)
game controllers
dongles w/ executeables (DVD player, etc.)
< 10 MHz
legacy
8/ 24+
DVD drive (game data / executeables)
FLASH ROM (bootloader + OS kernel)
Possible key path
Possible key path

Too many pins



secure hardware boundary

pentium CPU

133 MHz GTL+ 64/ 32+

NV2A northbridge + gfx

128/ 21+ SSTL-2 200 MHz DDR

SDRAM 64 MB

200 MHz DDR HyperT 8/ 2

100Base-T

USB

MCPX southbridge

secret boot ROM

IDE

key-locked hard disk (executeables, cached data, save games)

game controllers

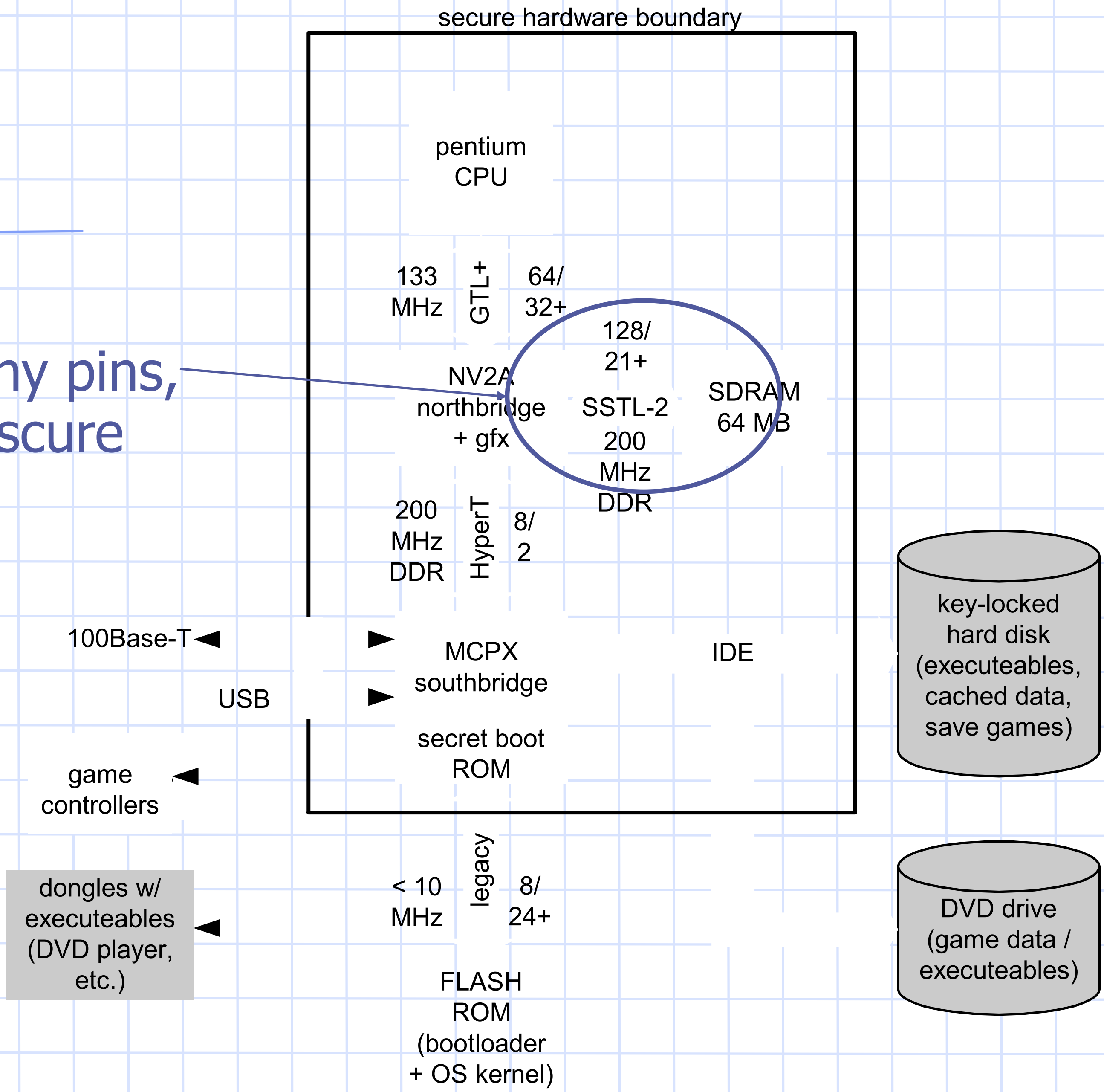
dongles w/ executeables (DVD player, etc.)

< 10 MHz legacy 8/ 24+

FLASH ROM (bootloader + OS kernel)

DVD drive (game data / executeables)

Too many pins, fast, obscure layout



secure hardware boundary

pentium CPU

133 MHz

GTL+

64/ 32+

NV2A northbridge + gfx

128/ 21+

SSTL-2

200 MHz DDR

SDRAM 64 MB

200 MHz DDR

HyperT

8/ 2

100Base-T

USB

MCPX southbridge

secret boot ROM

IDE

key-locked hard disk (executeables, cached data, save games)

game controllers

dongles w/ executeables (DVD player, etc.)

< 10 MHz

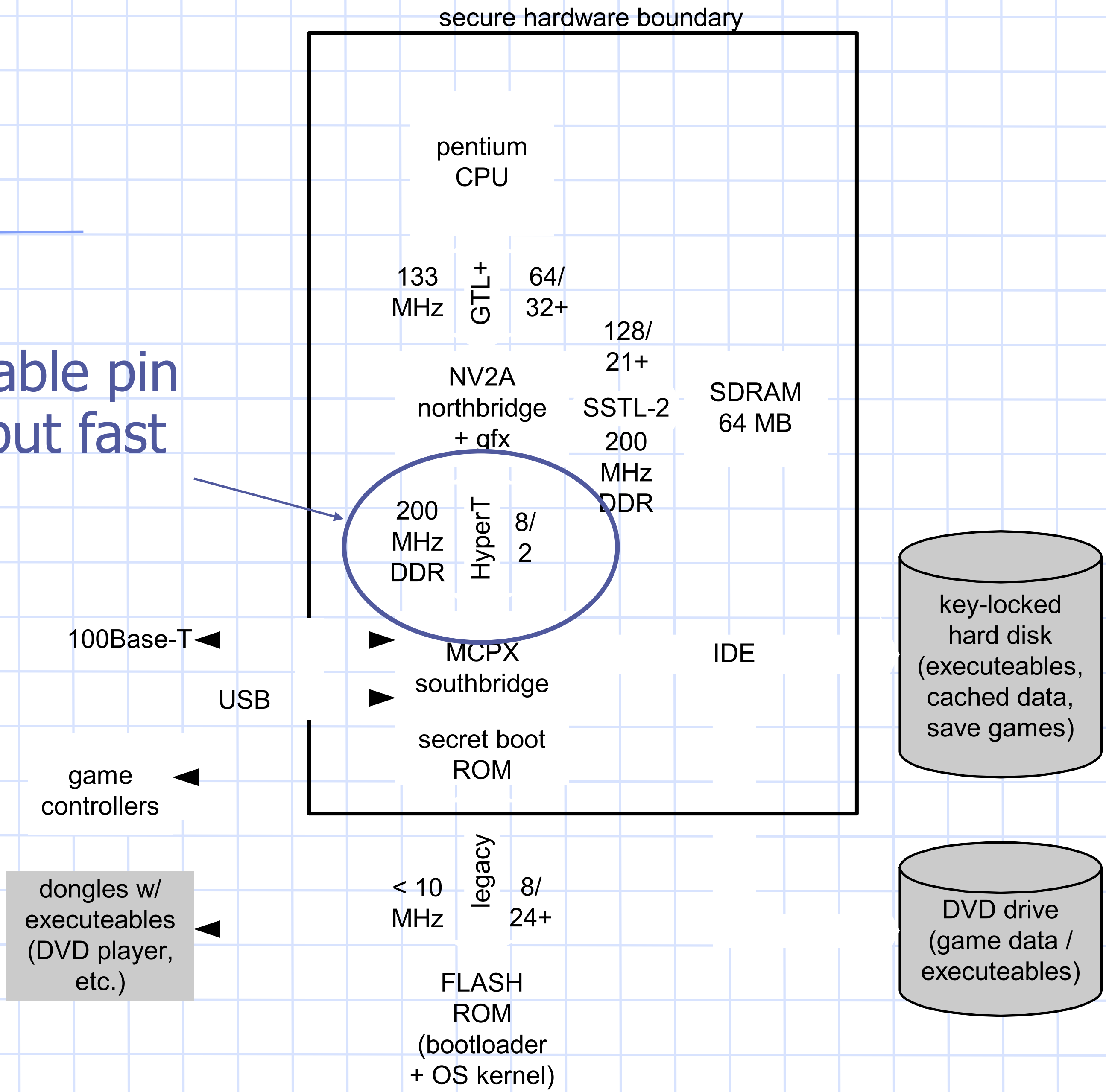
legacy

8/ 24+

FLASH ROM (bootloader + OS kernel)

DVD drive (game data / executeables)

Reasonable pin count, but fast



secure hardware boundary

pentium
CPU

133
MHz

GTL+

64/
32+

NV2A
northbridge
+ gfx

128/
21+

SSTL-2

200
MHz
DDR

SDRAM
64 MB

200
MHz
DDR

HyperT

8/
2

100Base-T

USB

MCPX
southbridge

IDE

key-locked
hard disk
(executeables,
cached data,
save games)

secret boot
ROM

game
controllers

dongles w/
executeables
(DVD player,
etc.)

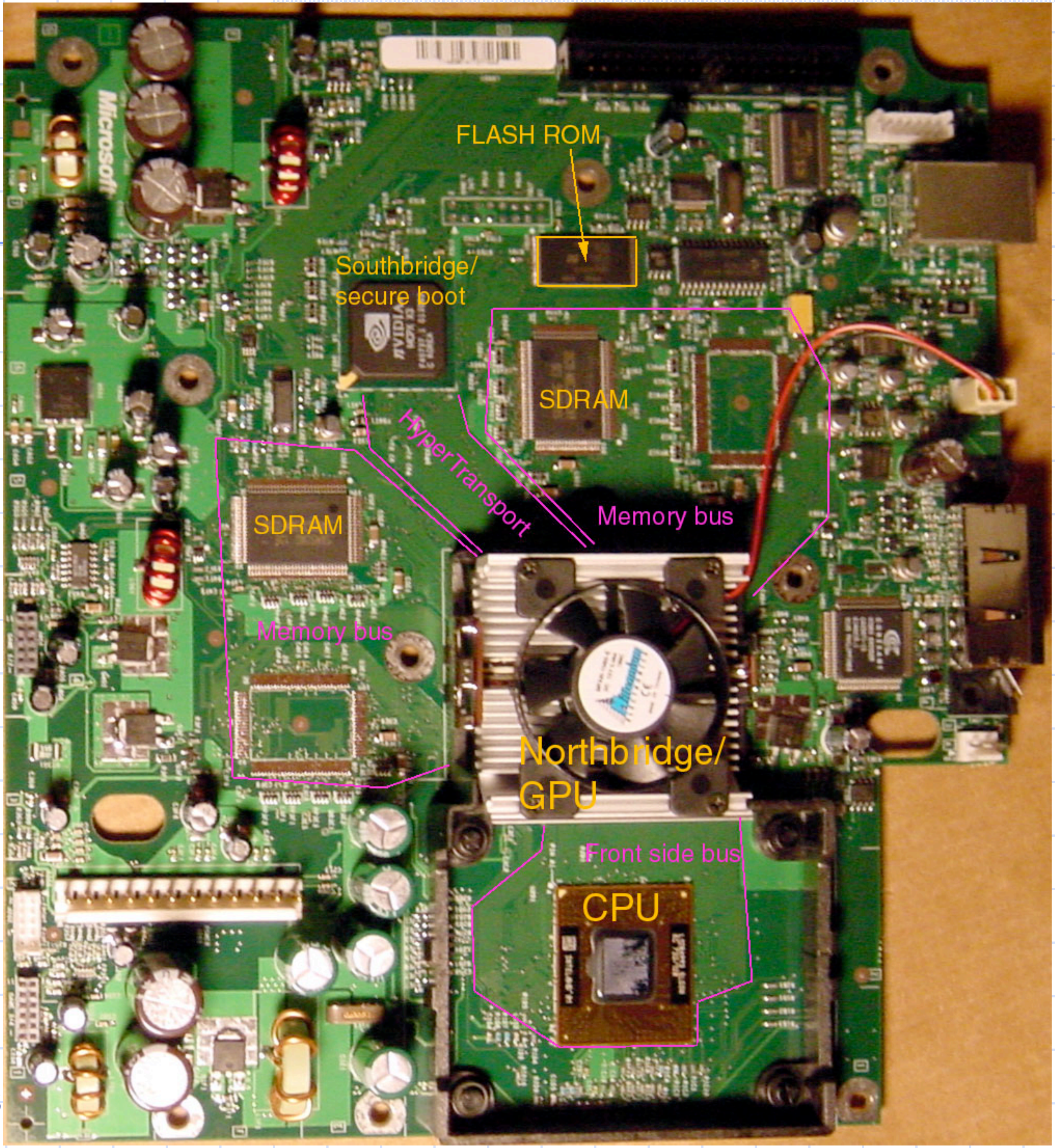
< 10
MHz

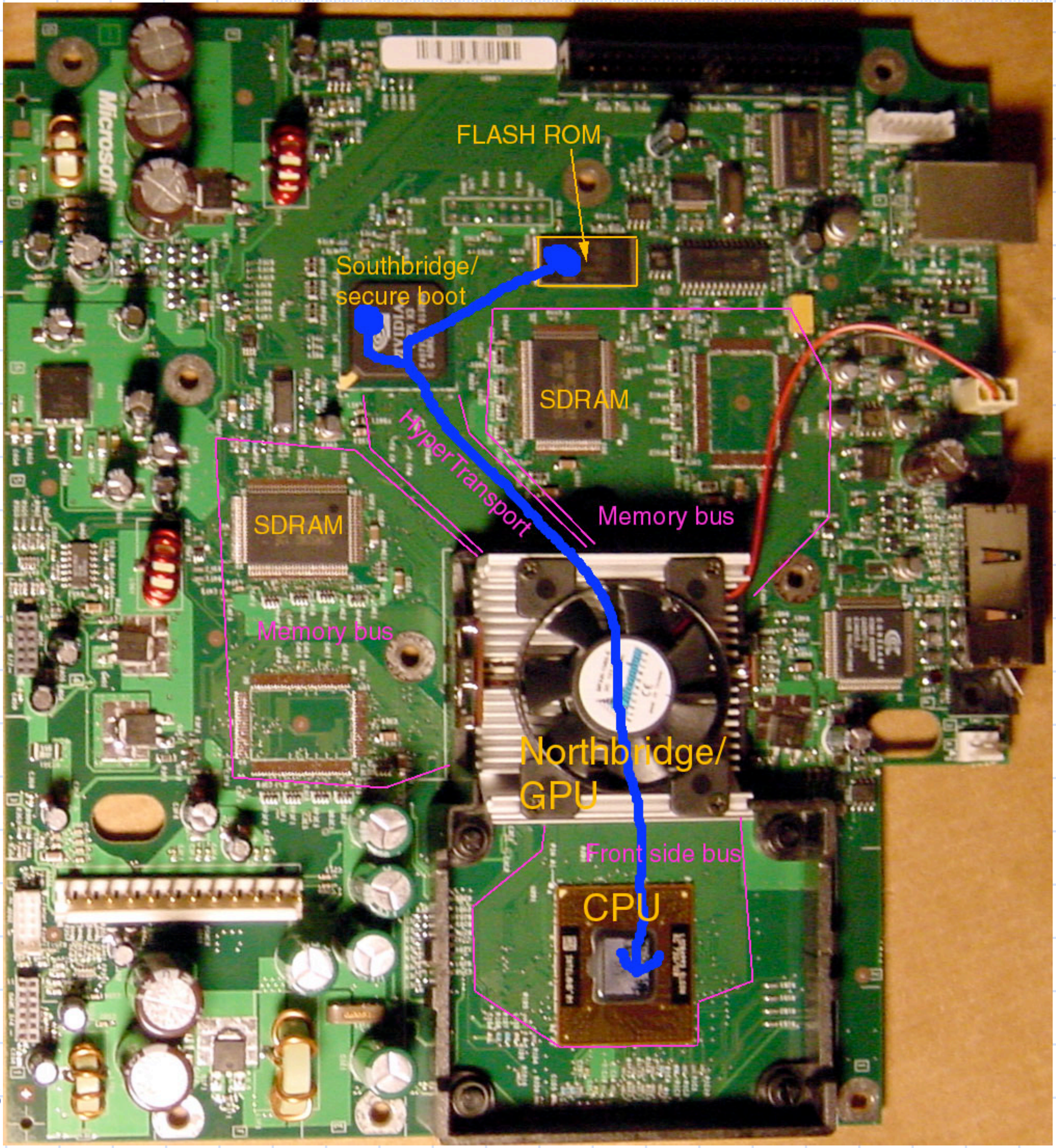
legacy

8/
24+

DVD drive
(game data /
executeables)

FLASH
ROM
(bootloader
+ OS kernel)

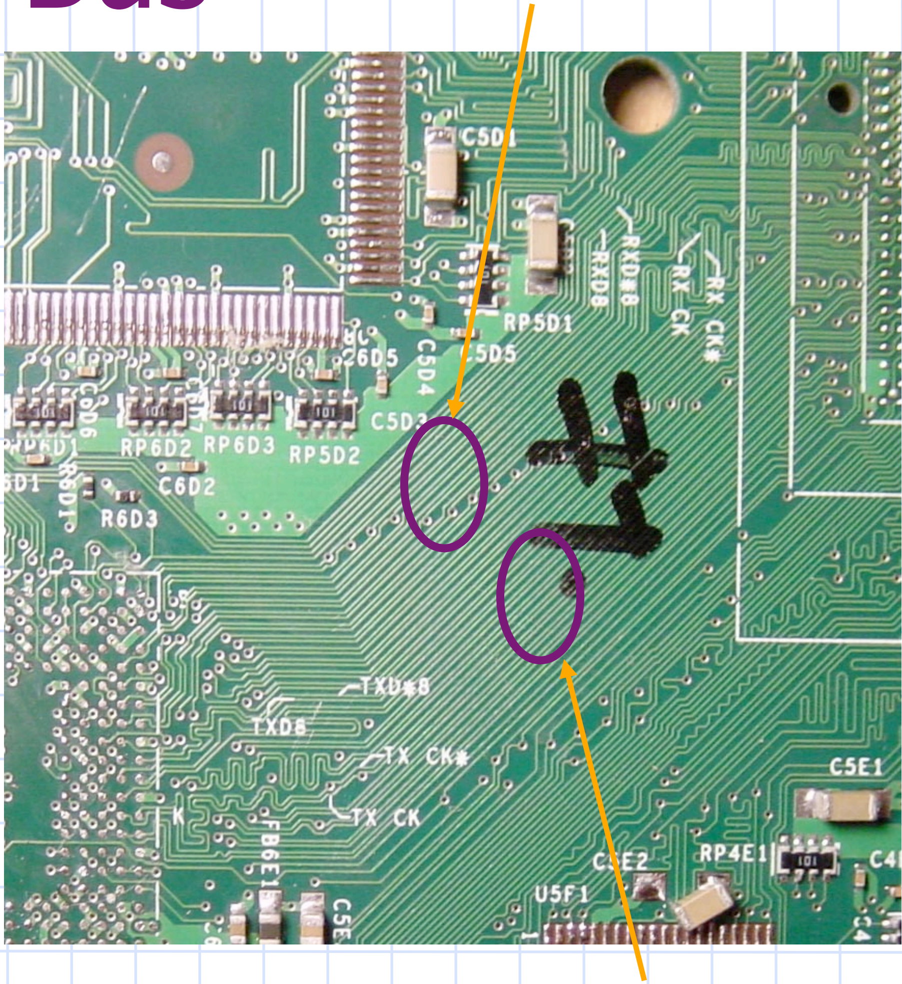

FLASH ROM
Southbridge/
secure boot
SDRAM
HyperTransport
Memory bus
SDRAM
Memory bus
Northbridge/
GPU
Front side bus
CPU


FLASH ROM
Southbridge/
secure boot
SDRAM
HyperTransport
Memory bus
SDRAM
Memory bus
Northbridge/
GPU
Front side bus
CPU

HyperTransport Bus

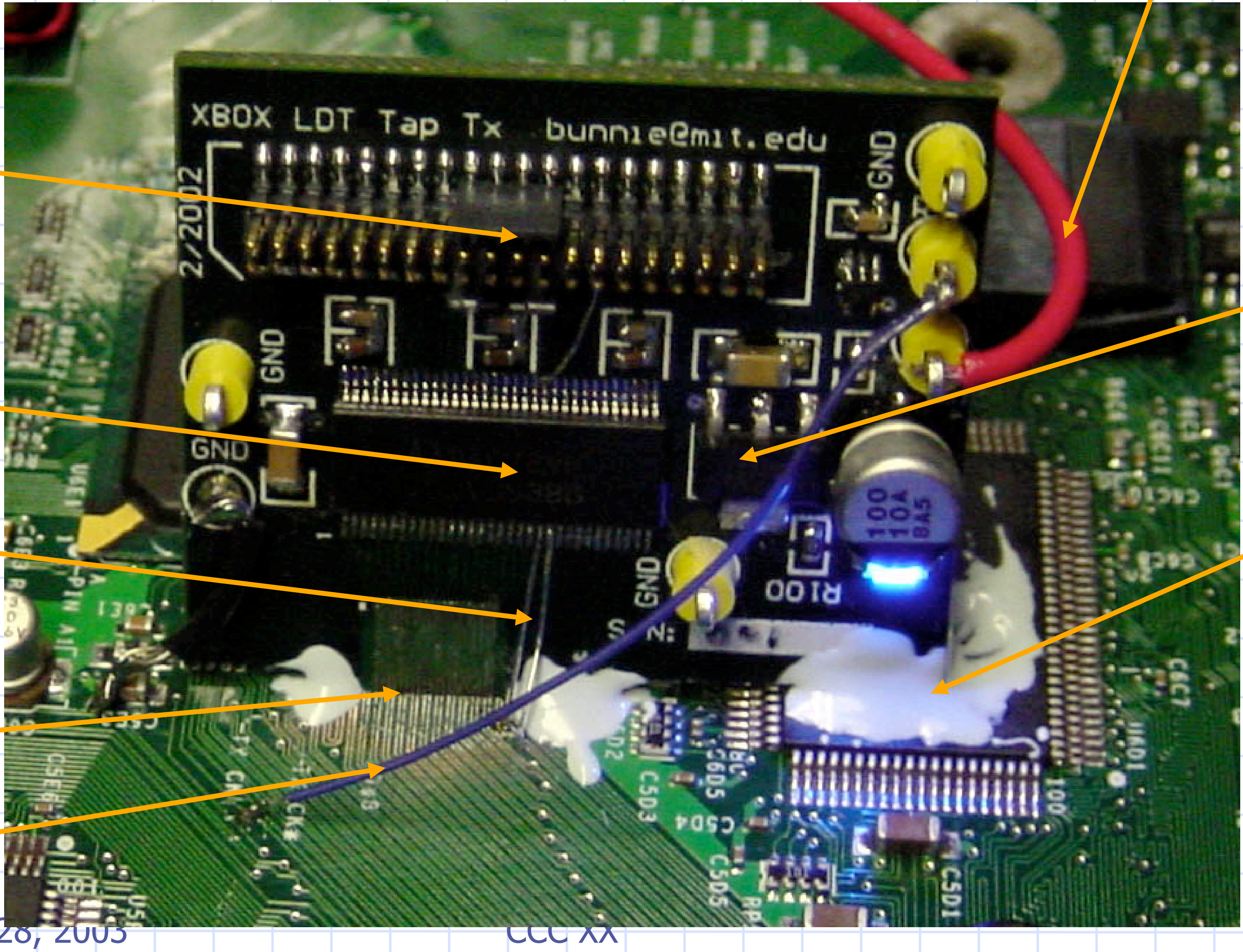
- Favorable board layout, pin count
  - Fabricate pitch-matched tap board
- High speed
  - Use high-end FPGA or logic analyzer

Rx bus



Tx bus

Custom Tap Board



+5V power in

CTT to FPGA

LVDS-TTL converter

+3.3V Local regulator

Epoxy in place

Last-minute signal pair

Pitch-matched HT connector

Reset signal

Tap Board

- Board adapts HyperTransport bus to existing hardware
  - Virtex-E FPGA board developed for my thesis
- Clean-sheet tap board would look different
  - Virtex-II FPGA directly on tap board
  - Would cost $50-$100 to fabricate

Analyzing the Bus

- Polarity and permutation of bus still unknown
  - Polarity determined using a known idle sequence (all 0's)
  - Permutation decoded by guided guesses
    - 1's count on a cache-line basis used to line up potential matches
    - Columns of bits permuted until all columns matched known patterns
    - Known patterns based on sections of the FLASH that were transmitted on the bus

Analyzing the Bus

- Traces of data collected, synchronized to power-on reset
- Ciphertext sorted from code by histogramming and eyeballing
- Data in traces organized by cache line
  - Code path was patched together using a disassembler and cache line groupings

Data Traces

Cycles since reset

Data on bus

Unaligned data

Jump instruction @ Boot vector

Code fetch

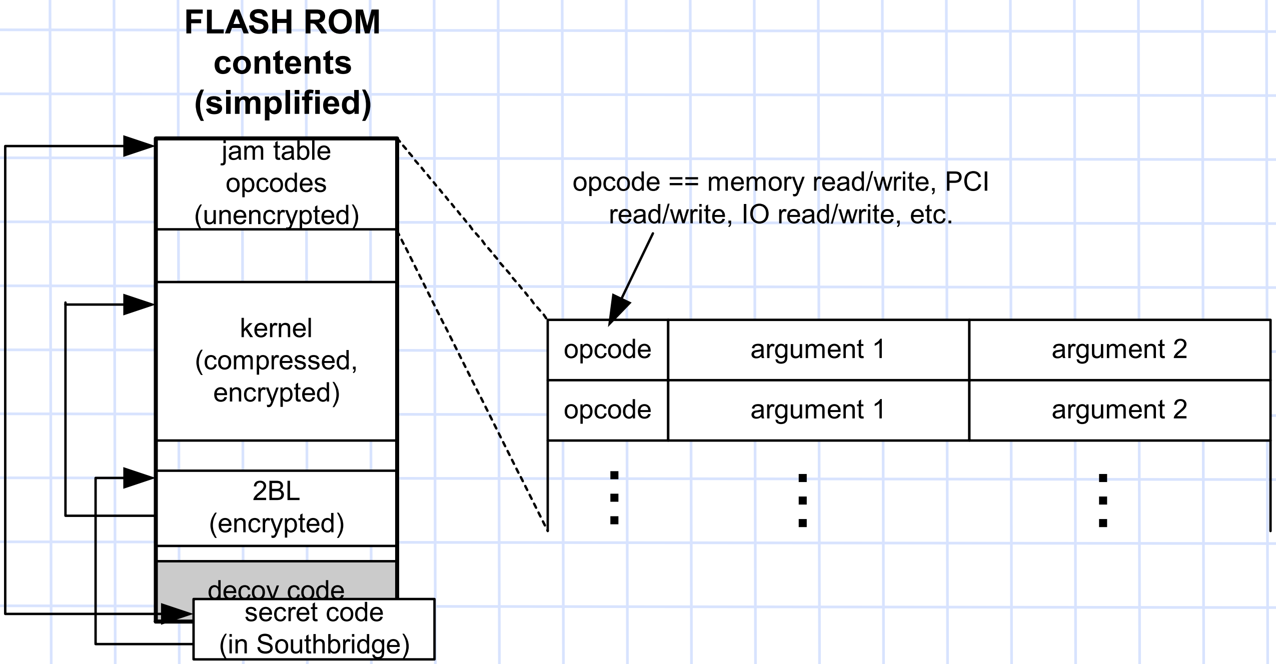
```
00000097 : FFFFFFFF ::: E : 000000FF
00000D5C : 090000FF ::: F : FFFFFF00
00000DE0 : 65D0162B ::: 1 : 00F707FF
00000E5D : 2D324633 ::: E : 09000000
00000EDA : 01010101 ::: 1 : 000000FF
00000F57 : 08080808 ::: E : 65D01600
00000FD4 : 01080000 ::: 1 : 0000002B
00001051 : 8A7CFCC8 ::: E : 2D324600
000010CE : 13022944 ::: 1 : 00000033
0000114B : 98490090 ::: E : 01010100
00001245 : FFFFFFFF ::: 1 : 00000001
000012C2 : FFFFFFFF ::: E : 08080800
00022526 : EBC68BFF ::: 1 : 00000008
00022527 : 1800D8FF ::: E : 01080000
00022528 : FFFF80C2 ::: E : 8A7CFC00
00022529 : 04B002EE ::: 1 : 000000C8
000226D5 : FFFF0000 ::: E : 13022900
000226D6 : 009BCF00 ::: 1 : 00000044
000226D7 : FFFF0000 ::: E : 98490000
000226D8 : 0093CF00 ::: 1 : 00000090
```

Piecing it Together

- Traces assembled into an image of the secure boot ROM
  - Secure boot ROM image contains
    - Jam table initialization
    - RC4 decryption routine
    - RC4/128 key
    - Magic number check

Summary

- Memory structure of the Xbox


FLASH ROM contents (simplified)
jam table opcodes (unencrypted)
kernel (compressed, encrypted)
2BL (encrypted)
decoy code
secret code (in Southbridge)
opcode == memory read/write, PCI read/write, IO read/write, etc.
opcode
argument 1
argument 2
opcode
argument 1
argument 2

Fragile Security

- All Xboxes used the same secret key
  - One key extraction applies to all boxes
  - Debug and test features on the Xbox motherboard enable easy ROM override
    - Easy to create, encrypt, and deploy mass quantities of untrusted hardware

Outline

- Background
- History of the first hardware hacks
- Summary of security
- Later hacks
- Future possibilities

Break for 5 Minutes

Outline

- Background
- History of the first hardware hacks
- Summary of security
- Later hacks
- Future possibilities

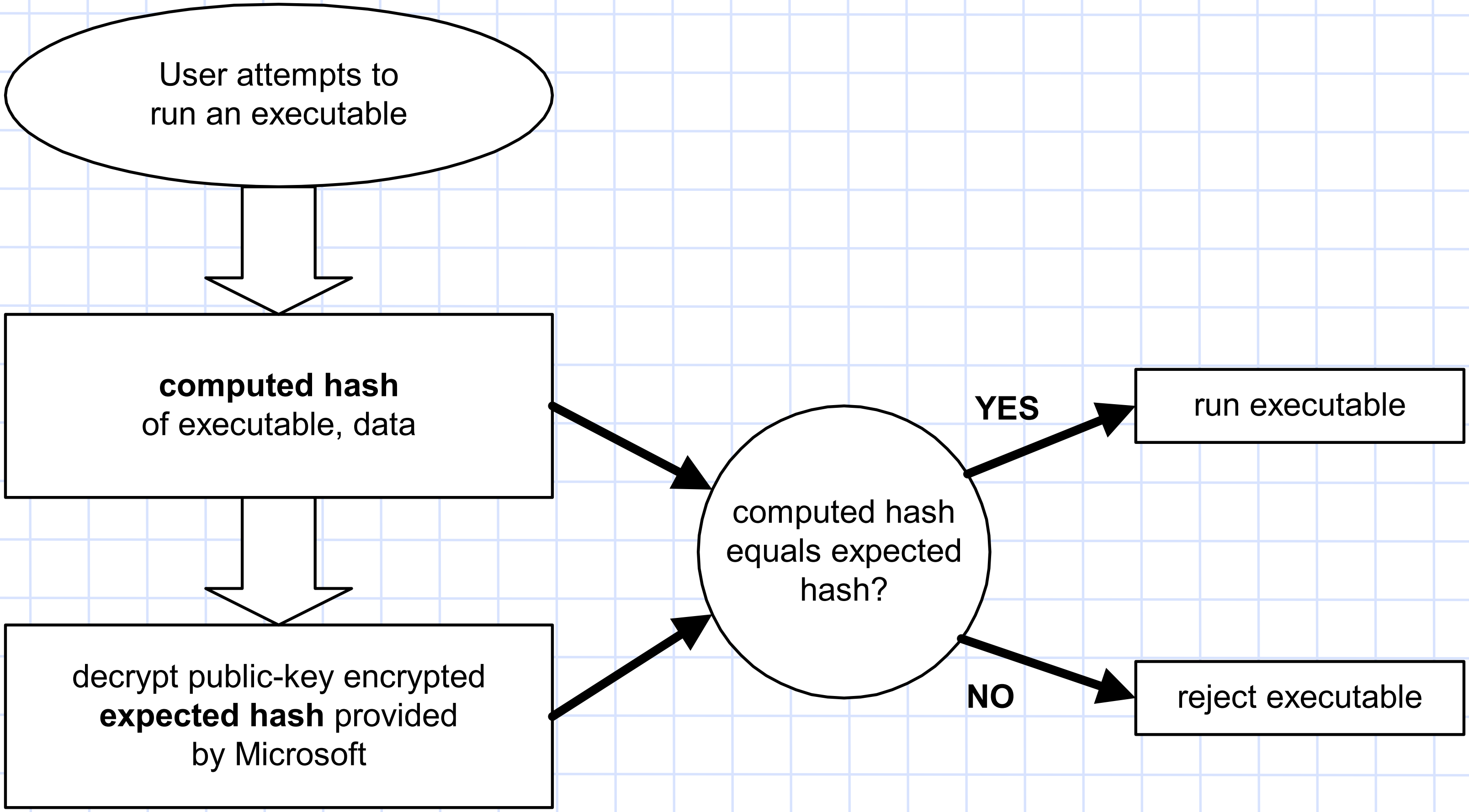
Xbox Security Review

- Xbox is a Trusted PC Platform
  - Comparable in spirit to Palladium™, TCPA
  - Hardware is trusted, all executables digitally signed and verified prior to execution
- Physical copy protection
  - 2-Layer DVD-9 format + block scrambling
  - 2-Layer DVDs are difficult to copy
- Encrypted network connections
  - No details available yet, Xbox Live not yet launched
- Minimal perimeter security, tamper evidence

Focus on Trust Mechanism

- Trustable hardware is a cornerstone of Xbox security
  - If hardware is compromised, there is no security

Why Trust is Required



User attempts to run an executable

computed hash of executable, data

decrypt public-key encrypted **expected hash** provided by Microsoft

computed hash equals expected hash?

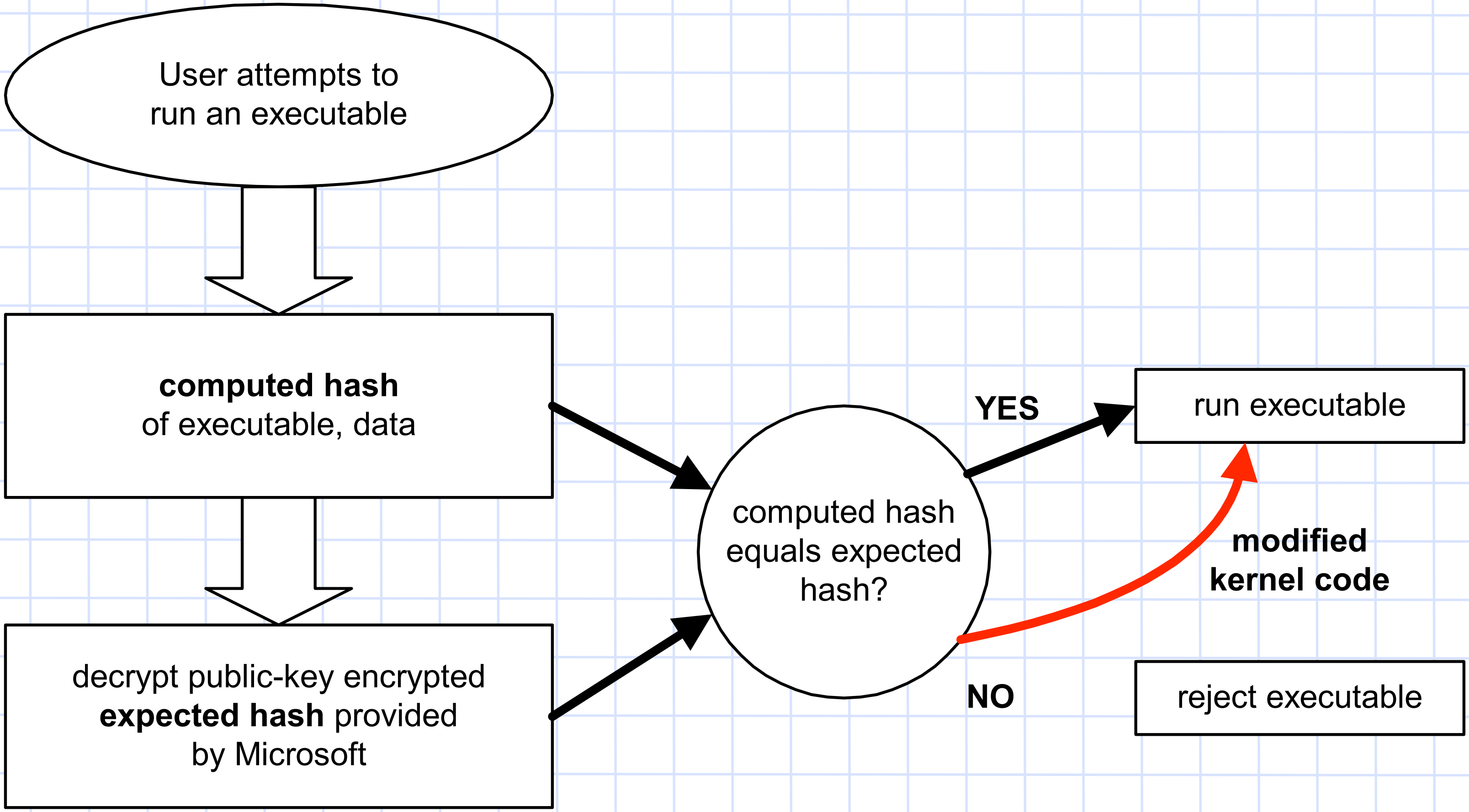
YES

run executable

NO

reject executable

Why Trust is Required



Establishing Trust

- Requirements
  - The program counter (PC) is always within a trusted code region, starting with the reset vector
  - All code and data is verified against signed hashes before being accepted
  - Code and hardware is free of bugs
    - i.e., buffer and segment overruns, protocol weaknesses
  - Hardware is inviolable
    - Intrusion detection at a minimum
    - Tamper resistance preferable

Establishing Trust

Microsoft does these

- Requirements
  - The program counter (PC) is always within a trusted code region, starting with the reset vector
  - All code and data is verified against signed hashes before being accepted
  - Code and hardware is free of bugs
    - i.e., buffer and segment overruns, protocol weaknesses
  - Hardware is inviolable
    - Intrusion detection at a minimum
    - Tamper resistance preferable

Establishing Trust

- Requirements
  - The program counter (PC) is always within a trusted code region, starting with the reset vector
  - All code and data is verified against signed hashes before being accepted
  - Code and hardware is free of bugs
    - i.e., buffer and segment overruns, protocol weaknesses
  - Hardware is inviolable
    - Intrusion detection at a minimum
    - Tamper resistance preferable

Attempts to do these

Establishing Trust

- Requirements
  - The program counter (PC) is always within a trusted code region, starting with the reset vector
  - All code and data is verified against signed hashes before being accepted
  - Code is free of bugs
    - i.e., buffer overruns, protocol weaknesses
  - Hardware is inviolable
    - Intrusion detection at a minimum
    - Tamper resistance preferable

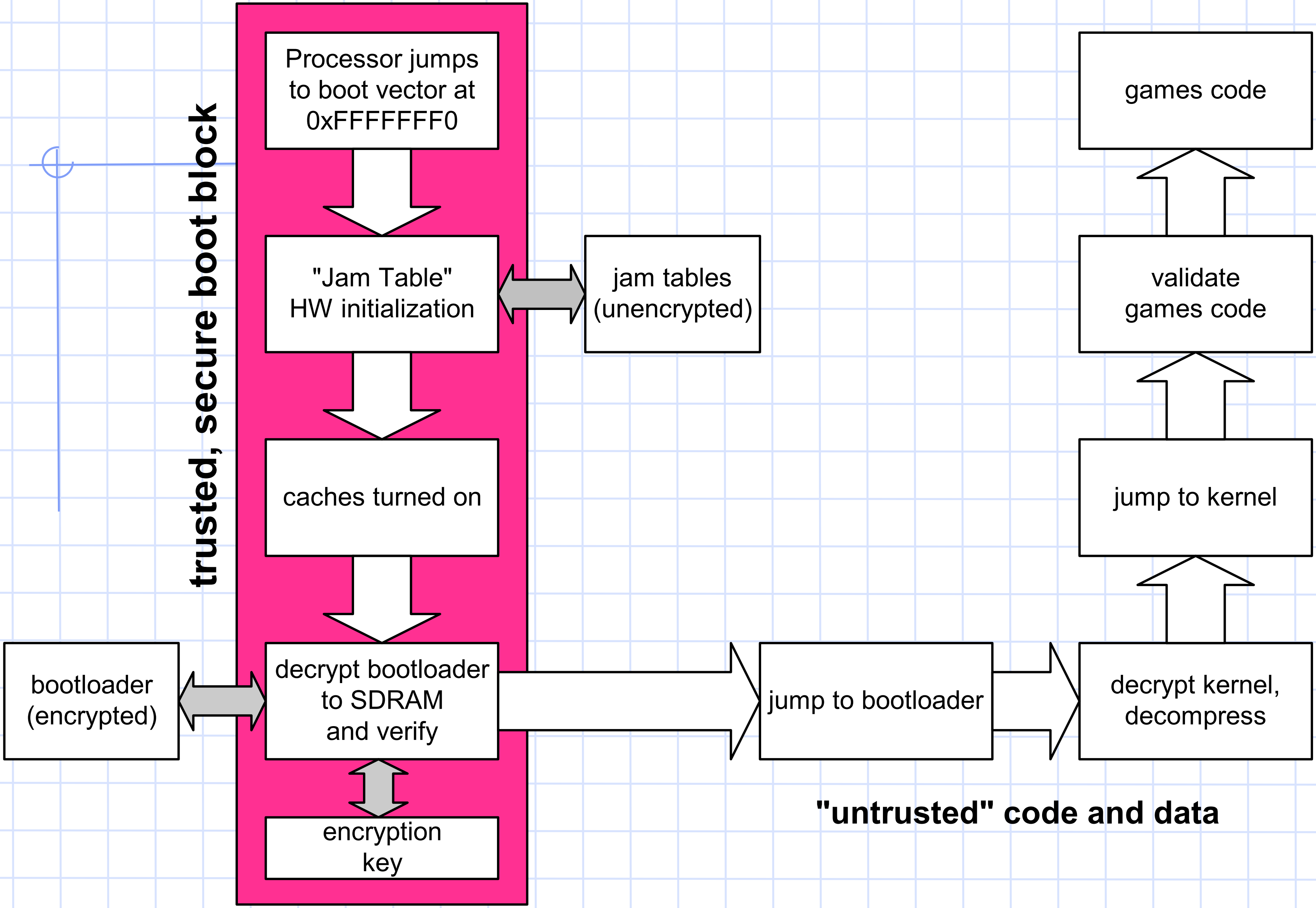
Fails to do this

Root of Trust

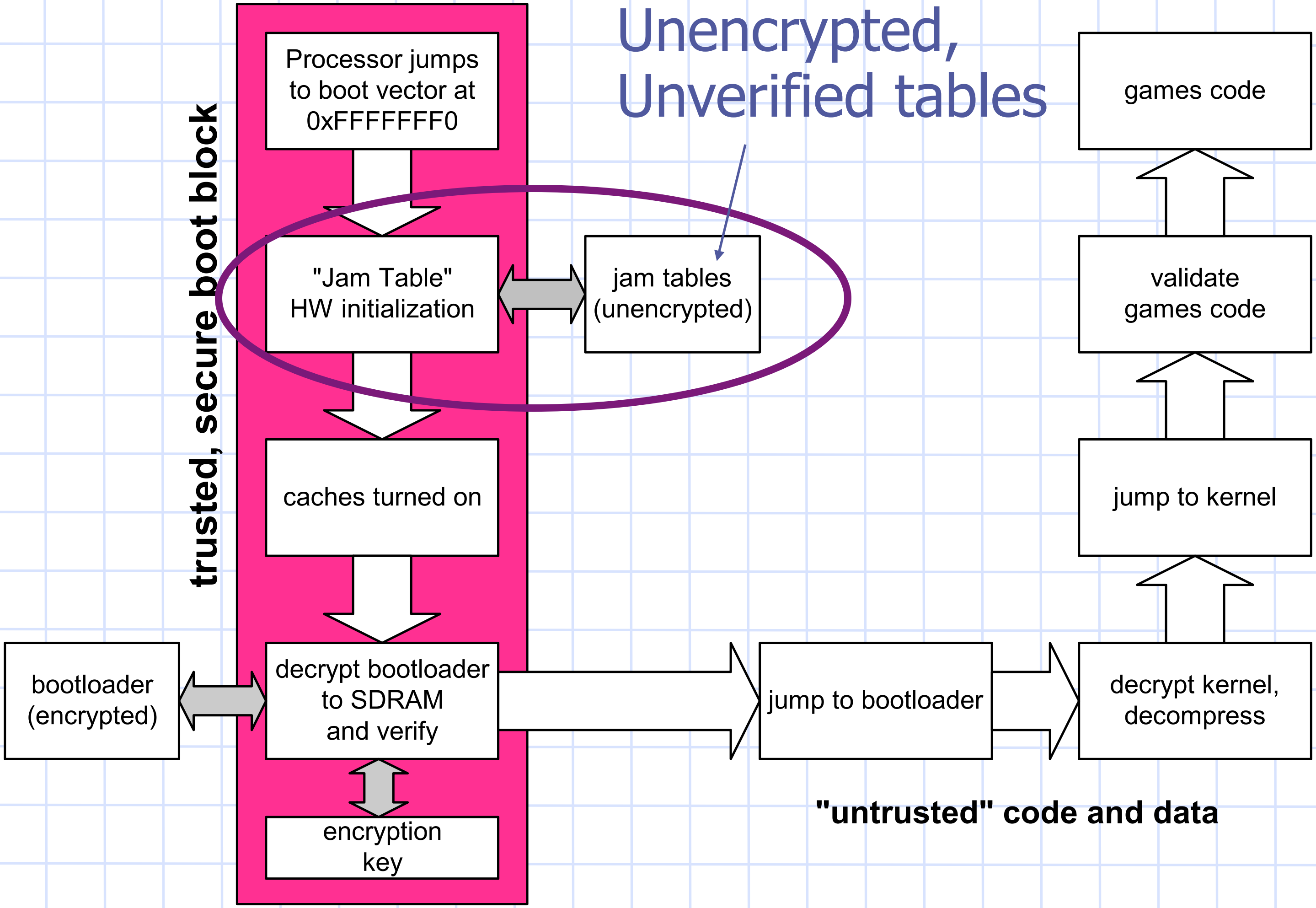
- Linear trust mechanism
  - Chain of trustable, verified code, starting with the secure boot block
- Secure boot block details
  - Reset vector/init code is contained in a tamper-resistant module
    - ROM overlay within the system peripherals ASIC (“southbridge” ASIC)
    - Southbridge ASIC implemented in 0.15μ, 6 or 7 layers of metal
      - Very hard to probe or modify

Transferring the Trust

- RC4/128 used to encrypt bootloader image
  - RC4/128 is a stream cipher
    - A ciphertext modification will corrupt the remainder of the plaintext stream
  - Simple "magic number" at the end of the bootloader image, checked to verify integrity
- So long as the RC4/128 key is secret, attackers are unlikely to generate a valid false bootloader image
  - Secondary bootloader continues to transfer trust through verification of digitally signed binaries


trusted, secure boot block
Processor jumps to boot vector at 0xFFFFFFF0
"Jam Table" HW initialization
jam tables (unencrypted)
caches turned on
bootloader (encrypted)
decrypt bootloader to SDRAM and verify
encryption key
jump to bootloader
decrypt kernel, decompress
jump to kernel
validate games code
games code
"untrusted" code and data

Backdoors Galore


trusted, secure boot block
Processor jumps to boot vector at 0xFFFFFFF0
"Jam Table" HW initialization
jam tables (unencrypted)
Unencrypted, Unverified tables
caches turned on
bootloader (encrypted)
decrypt bootloader to SDRAM and verify
encryption key
jump to bootloader
decrypt kernel, decompress
jump to kernel
validate games code
games code
"untrusted" code and data

Outline

- Background
- History of the first hardware hacks
- Summary of security
- Later hacks
- Future possibilities

Jamtable Interpreter

- What it is
  - Bytecode interpreter
  - Orchestrates dependencies and decisions required for machine initialization
- What it can do
  - Reads and writes to PCI, memory, I/O space
  - Conditional jumps, indirect addressing

Jamtable Opcodes

0x02	PEEK	ACC := MEM[OP1]
0x03	POKE	MEM[OP1] := OP2
0x04	POKEPCI	PCICONF[OP1] := OP2
0x05	PEEKPCI	ACC := PCICONF[OP1]
0x06	AND/OR	ACC := (ACC & OP1) | OP2
0x07	(prefix)	execute the instruction code in OP1 with OP1 := OP2, OP2 := ACC
0x08	BNE	IF ACC = OP1 THEN PC := PC + OP2
0x09	BRA	PC := PC + OP2
0x10	AND/OR ACC2	*(unused/defunct)* ACC2 := (ACC2 & OP1) | OP2
0x11	OUTB	PORT[OP1] := OP2
0x12	INB	ACC := PORT(OP1)
0xEE	END	

Jamtable Attacks (visor)

- Jamtables are unencrypted and unverified
  - Can perform attacks without crypto
  - Two-phase soft-reset attacks to read out plaintext
    - Allow machine to power up normally once, then soft reset with a new jam table that copies code to an insecure location (courtesy visor)

Jamtable Attacks II (visor)

- Jamtable weakness + hardware bugs allows program counter to be seized
  - Secure boot block jumps to 0xFFFF FFFA when a bad ciphertext image is encountered
  - PC will roll over from 0xFFFF FFFF to 0x0000 0000 without an exception
  - 0x0000 0000 is in SDRAM memory
  - Use jamtable to write at 0x0000 0000 a jump instruction to an insecure FLASH region, and corrupt ciphertext image to sieze the PC
  - Courtesy visor

Implications of Visor Attacks

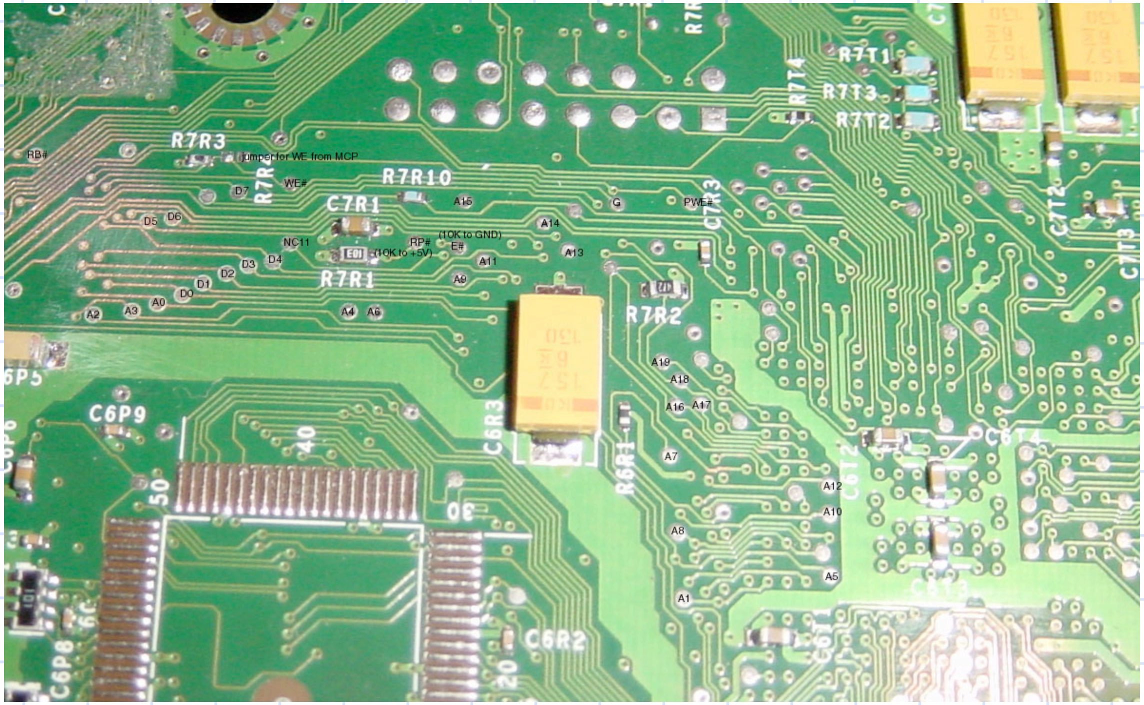
- A crypto-free way of bypassing the secure boot ROM
  - Allows for Linux to be installed without risk of exposing MSFT proprietary code in the plain
  - More “legal” than key extraction approach
  - Method of choice

Alternate Firmware Devices

- Also referred to as the “modchip”
  - Significantly, AFD’s come with no code, making them much easier/more legal to sell and trade
- Two major approaches
  - Direct FLASH override
  - LPC FLASH override

Direct FLASH Override

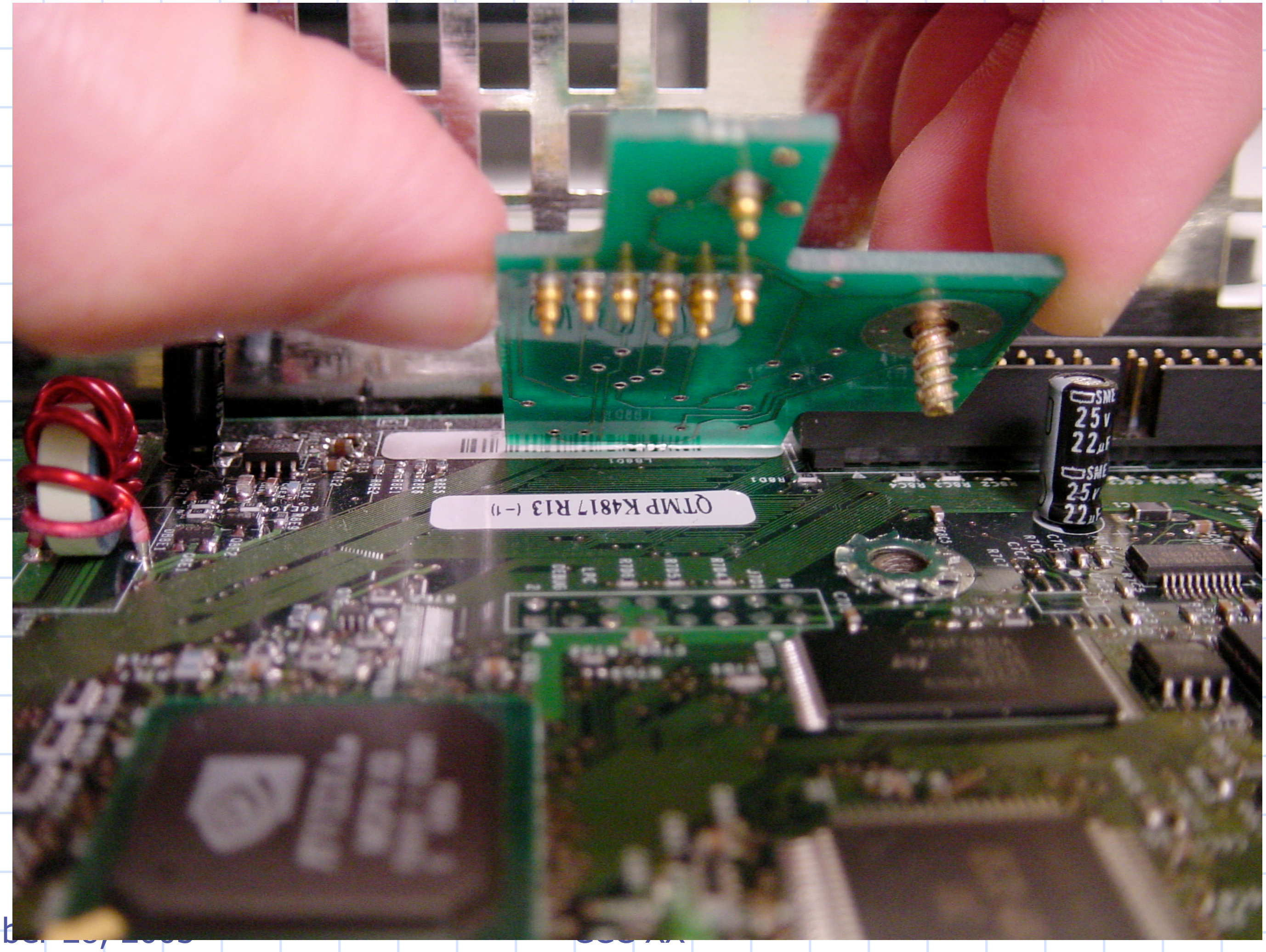
- A ROM chip is soldered, pin for pin, into the Xbox that supplants the FLASH chip



LPC FLASH override

- Corrupting the FLASH causes the Xbox to fall back to the LPC bus for an alternate ROM image
  - "corruption" is as simple as tying D0 to ground
- LPC bus is contact-probeable
  - Can use solderless "pogo-pin" methods
  - Makes it a very popular, easy to install method that is relatively risk-free

LPC FLASH Override



Why Does It Exist?

- LPC port seems to be essential to the Xbox
  - Later versions of the Xbox have changed available signals to thwart modders, but have not done away with the port
- LPC is a great debug/diagnostic port
  - Use LPC to program factory-blank ROMs
  - Use LPC to diagnose production rejects

MSFT Countermeasures

- nVidia has a terrible quarter—I feel terrible!

> “What we said about Xbox was that we reached a volume discount milestone, further reducing the margins. And that we will be taking an inventory write off in Q2 related to the amount of Xbox MCPs that were made obsolete when MSFT transitioned to a new security code (by way of the MIT hacker) and excess in nForce chipsets that we built in anticipation of higher demand of Athlonbased PCs.”
>
> - Derek Perez, PR Director nVidia

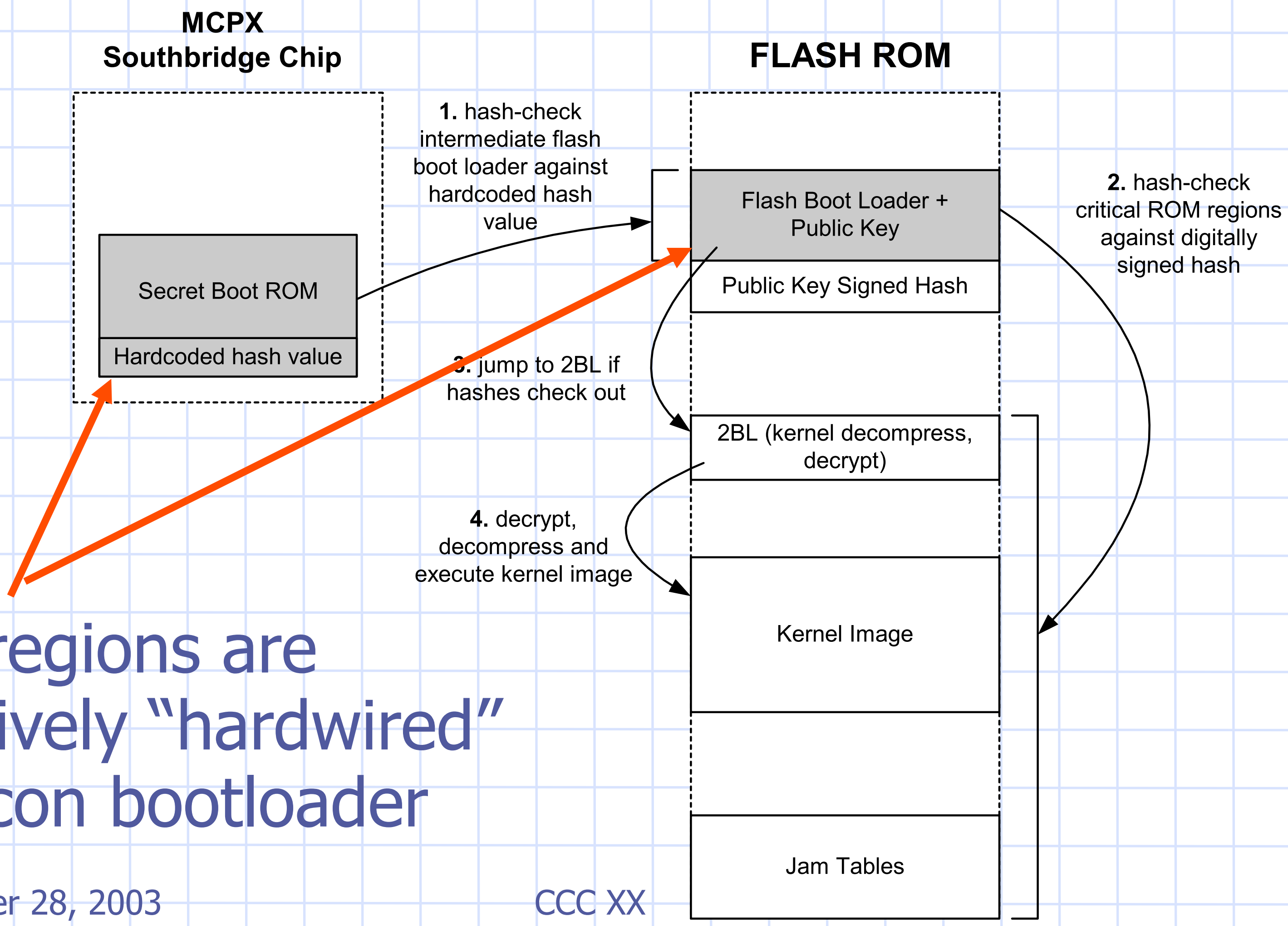
Version 1.1

- A new set of Xbox hackers take over here
  - Xbox-Linux team has a go at the security

The New Security

- Andy Green extracted the new MCPX contents through a back door discovered by Jeff Mears
- Analysis revealed that the Xbox’s FLASH memory was verified using a hash function

New Security Algorithm



MCPX
Southbridge Chip

Secret Boot ROM

Hardcoded hash value

FLASH ROM

Flash Boot Loader +
Public Key

Public Key Signed Hash

2BL (kernel decompress, decrypt)

Kernel Image

Jam Tables

1. hash-check intermediate flash boot loader against hardcoded hash value

2. hash-check critical ROM regions against digitally signed hash

3. jump to 2BL if hashes check out

4. decrypt, decompress and execute kernel image

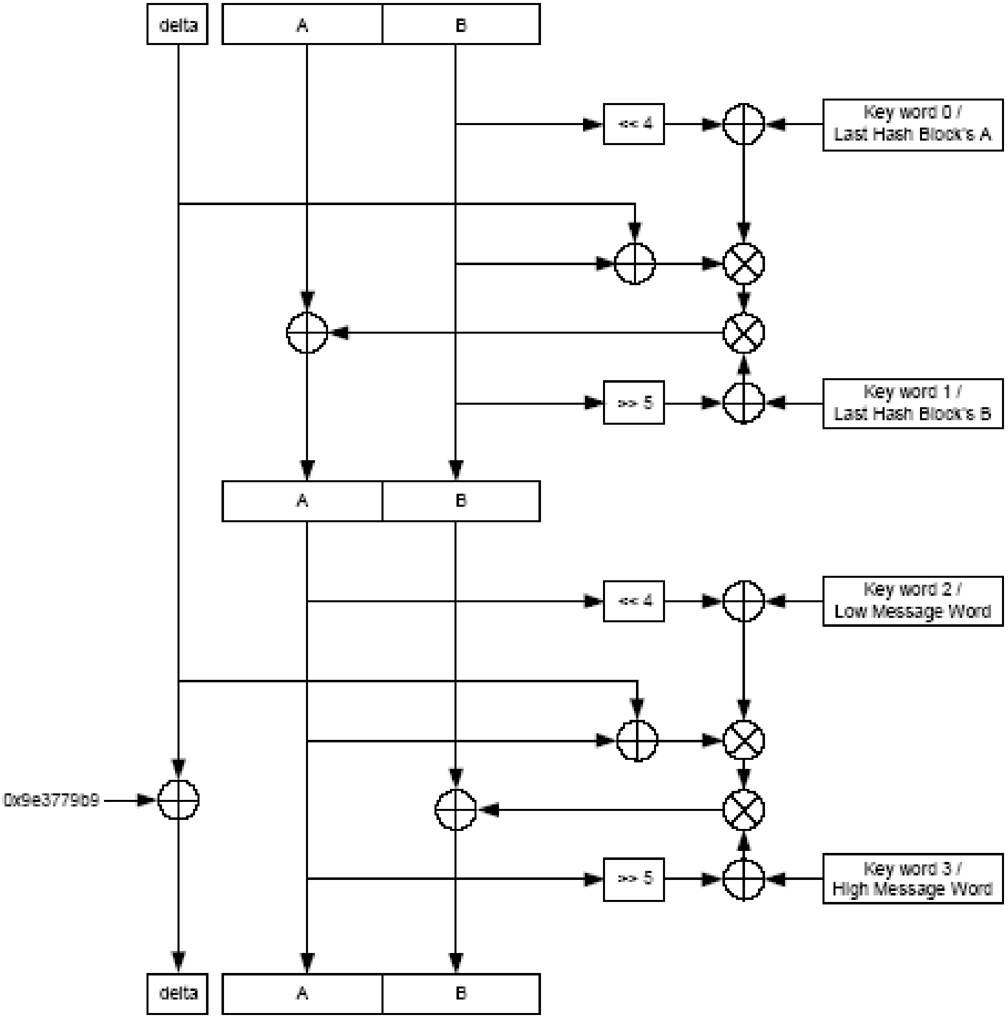
Gray regions are Effectively “hardwired” to silicon bootloader

Theoretically…

- Such a scheme is bullet-proof
  - Given no back doors
  - Given strong crypto algorithms
- …but even known back doors were left open
  - Enabled MCPX extraction
  - But does not enable the generic boot capability!

Hash Algorithm

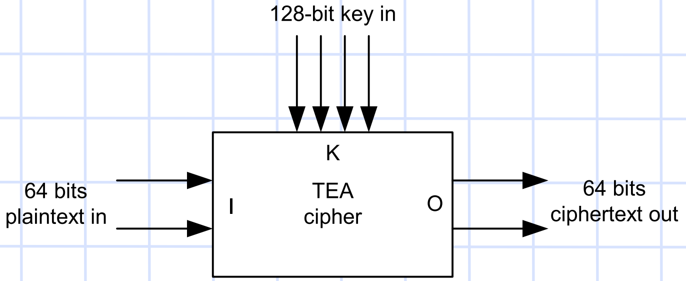
- TEA = Tiny Encryption Algorithm
  - Fiestel cipher
  - 32 rounds
  - 128-bit key
  - 64-bit datapath
  - Shift, xor and mod-add only


delta
A
B
<< 4
Key word 0 / Last Hash Block's A
>> 5
Key word 1 / Last Hash Block's B
A
B
<< 4
Key word 2 / Low Message Word
0x9e3779b9
>> 5
Key word 3 / High Message Word
delta
A
B

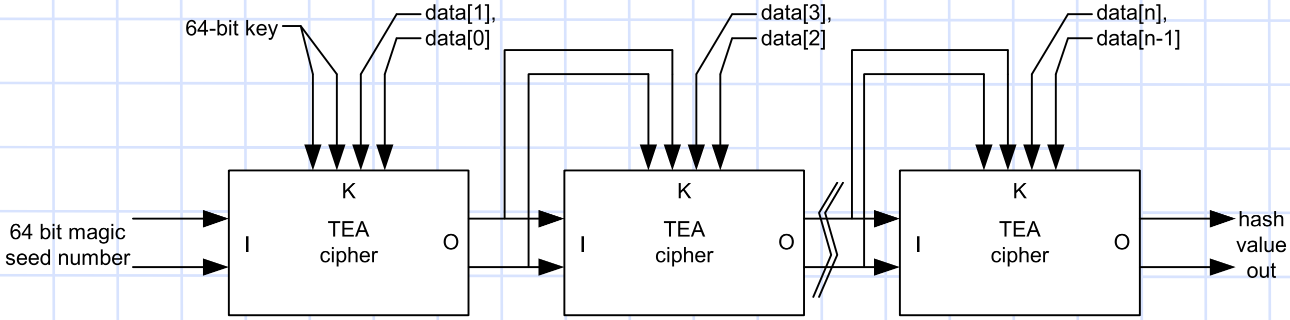
Hash Algorithm

- Operate cipher in chaining mode for hash

TEA in a cipher application



TEA used as a hash function



Known Weaknesses!

- A paper[1] in 1996 revealed a related-key weakness
  - Bits 31, 63 and 95, 127 of key can be simultaneously inverted and produce the same result for any input
- This is **not** good for a hash function
  - Extremely easy to generate such collisions

[1] "Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and triple-DES". John Kelsey, Bruce Schneier, and David Wagner. CRYPTO 1996.

A Stroke of Genius

- Discovered:
  - A long jump instruction early in the verified code
  - The argument of the jump could be manipulated to jump to SDRAM!
    - Manipulation yields same hash code so it passes the MCPX hash check
    - Jam tables are checked only by code executed downstream from the hash check, so SDRAM can be seeded with instructions à la Visor

Yet another Hack

- Public key replacement
  - Discovered by Franz Lehner
  - The verified code includes an RSA key used to verify that the kernel image and ROM have not been changed
  - TEA allows pairs of properly aligned bits to be complemented
  - This includes the public key
  - We can “sign” new kernels as if we were Microsoft by changing the key

RSA with prime numbers

- In normal RSA, the modulus “n” is the product of 2 prime numbers (p and q)
  - The number Φ(n) is (p-1)(q-1)
- RSA works when n is prime too
  - The number Φ(n) is p-1 then
  - RSA still works
  - Insecure, but unimportant

RSA in the Boot Loader

- The RSA key used to verify the kernel signature in 1.1 is hashed by TEA
- TEA flaw allows bits 31 and 63 of any 64 bits to be simultaneously flipped
- We can change bit pairs in the RSA key
  - Change pairs until the key is prime
  - By the Prime Number Theorem, 1 in 268 2048-bit numbers are prime on average
  - About 2^{20} possible ways to flip bit pairs
    - Easy to find such a prime number
- With this, we can sign our own kernel

Remarkable Timing

- Total time to break security was about three days
  - Probably not worth the pain and suffering applied to nVidia

Outline

- Background
- History of the first hardware hacks
- Summary of security
- Later hacks
- Future possibilities

What MS Could Have Done

- Avoid symmetric ciphers in this scenario
  - Difficult to guarantee secrecy of key
  - Cost of ASIC mask sets, lead time make key rotation expensive and difficult
- Use hashes to verify **all** code and data regions
  - Complex protocols such as x86/PC initialization are difficult to secure
  - Requires a larger piece of code

Alternative Solution

- Use digital signatures to verify the FLASH ROM contents
  - Grow ROM size
  - Store signature in off-chip EEPROM, key in ASIC
  - Users cannot run false code without signer's private key
  - Does not prevent plaintext snooping
  - Can be defeated with a bus override attack
    - A set of precisely timed pulses on the HyperTransport bus can alter the reset vector

Bus Override Attack

Cycles since reset

Data on bus

Jump instruction @ Boot vector

```
00000097 : FFFFFFFF ::: E : 000000FF
00000D5C : 090000FF ::: F : FFFFFF00
00000DE0 : 65D0162B ::: 1 : 00F707FF
00000E5D : 2D324633 ::: E : 09000000
00000EDA : 01010101 ::: 1 : 000000FF
00000F57 : 08080808 ::: E : 65D01600
00000FD4 : 01080000 ::: 1 : 0000002B
00001051 : 8A7CFCC8 ::: E : 2D324600
000010CE : 13022944 ::: 1 : 00000033
0000114B : 98490090 ::: E : 01010100
00001245 : FFFFFFFF ::: 1 : 00000001
000012C2 : FFFFFFFF ::: E : 08080800
00022526 : EBC68BFF ::: 1 : 00000008
00022527 : 1800D8FF ::: E : 01080000
00022528 : FFFF80C2 ::: E : 8A7CFC00
00022529 : 04B002EE ::: 1 : 000000C8
000226D5 : FFFF0000 ::: E : 13022900
000226D6 : 009BCF00 ::: 1 : 00000044
000226D7 : FFFF0000 ::: E : 98490000
000226D8 : 0093CF00 ::: 1 : 00000090
```

Bus Override Attack

Data on bus

Cycles since reset

```
00000097 : FFFFFFFF ::: E : 000000FF
00000D5C : 090000FF ::: F : FFFFFF00
00000DE0 : 65D0162B ::: 1 : 00F707FF
00000E5D : 2D324633 ::: E : 09000000
00000EDA : 01010101 ::: 1 : 000000FF
00000F57 : 08080808 ::: E : 65D01600
00000FD4 : 01080000 ::: 1 : 0000002B
00001051 : 8A7CFCC8 ::: E : 2D324600
000010CE : 13022944 ::: 1 : 00000033
0000114B : 98490090 ::: E : 01010100
00001245 : FFFFFFFF ::: 1 : 00000001
000012C2 : FFFFFFFF ::: E : 08080800
00022526 : E9JMPDST ::: 1 : 00000008
00022527 : 1800D8FF ::: E : 01080000
00022528 : FFFF80C2 ::: E : 8A7CFC00
00022529 : 04B002EE ::: 1 : 000000C8
000226D5 : FFFF0000 ::: E : 13022900
000226D6 : 009BCF00 ::: 1 : 00000044
000226D7 : FFFF0000 ::: E : 98490000
000226D8 : 0093CF00 ::: 1 : 00000090
```

Override cycle 22526 with jump opcode to insecure code space

Alternative Solution, Cont’d

- Use digital signatures to verify the FLASH ROM contents
  - Can be defeated with a snoop & modify memory
    - Most effective in a PC using standard memory sockets
    - Present trust introspection routines with benign code images
    - Present malicious memory image at other times
    - Also use to snoop and extract plaintexts
    - Snoop-RAM can be fairly inexpensive to manufacture
    - Inspired by entries about Palladium in Seth Schoen’s online diary

Q&A

- Questions?

Thank you for your attention.